Framework and Approach for Managing Software Process Evolution in EPOS

Minh N. Nguyen

Department of Computer and Information Science
Norwegian University of Science and Technology

May 22, 1997
Abstract

Software processes are characterized as being complex, hard to manage and evolve rapidly. This dynamic and unpredictable nature of software process evolution is a major cause for most software projects are running late, exceeding planned budget or delivering poor quality products. The ability to anticipate process evolution and then effectively react to it has therefore been recognized as a characteristic feature of successful projects. Such recognition leads to a growing interest in software process technology, represented by a large number of Process-centered Software Engineering Environments prototypes (PSEEs) focusing on formal modeling language and process enactment. Meanwhile, theoretical research on software process improvement (SPI) and other disciplines such as software measurement, project management and experience reuse has resulted in many valuable achievements. However, there is still little effort spent in incorporating theoretical SPI models into PSEEs to better coordinate synergy effect towards software industry. In addition, concrete benefits of PSEEs for software practitioners have not been properly validated by industrial case studies or experiments, neither for process modeling nor for process enactment. Our motivation is therefore considered as threefold: i) to define a categorization framework and associated approaches to manage process evolution; ii) to integrate our work on process evolution into our PSEE prototype together with other existing theoretical work in other disciplines; iii) to validate our integrated PSEE with both process examples and a full-scaled case study against a real industrial setting. The three motivations are elaborated as follows.

Management process evolution

First, this thesis proposes a flexible framework to categorize process evolution during software projects. Such a categorization framework provides a means to better understand, analyze and then recognize most critical or frequent patterns of process evolution. The categorization framework for process evolution is based on three abstraction levels: dimension, aspect and category. The dimension specifies pertinent questions such as where evolution is originated; why evolution is caused; what entities are affected; when evolution is detected/corrected; how evolution is handled/acted; and by-whom evolution is corrected. The framework aspect and category is then constructed to accommodate a particular situation in organization. An unplanned evolution occurrence often implies a violation of project schedule and a corrective cost. Such additional cost is recorded together with the evolution occurrence and a project context in which the evolution occurs. The project context is characterized by a set of profile measures (or indicators); original estimates and actual outcomes. The project profile indicators are used to determine similarities between projects. In that way, we can recognize empirical relations between such project profile indicators and critical or frequent evolution patterns.

Further, the thesis proposes an empirical evolution planning approach which is a combination of traditional process planning technology, process measurement and experience reuse. The approach which mainly relies on recorded evolution experiences from completed projects, has three major steps: i) establishing relevant baselines for future projection; ii) supporting initial planning based on historical data from completed project with best-matching profiles; iii) supporting iterative planning to deal with unanticipated evolution and estimate its impact. The approach analyzes the recorded evolution data and helps to make precise estimates. Process evolution patterns which likely occur in a particular project, are initially anticipated by incorporating suitable contingency factor in the estimates. During project execution, the approach helps to predict consequences of unplanned process evolution. We also
propose an approach to address unplanned process evolution by integrating features from project management into a PSEE. Such features manipulate the activity network layout or revise activity schedule and resource within a context of a PSEE.

Integration and implementation in a PSEE

The proposed categorization framework and the associated approaches are then implemented and integrated in EPOS being a PSEE prototype. EPOS consists of a versioned database EPOSDB, including among other nested transactions, client server architecture and flexible cooperation supports. EPOS-PM, residing on top of EPOSDB, provides a reflective, object-oriented language for modeling software processes and a set of process tools for browsing, creating, instantiating, executing and evolving the process models. The existing data structure in EPOS is extended to represent evolution data and other relevant experience information. An experience database is thus designed and implemented together with tools to characterize project; record/categorize evolution occurrences; analyze collected evolution data; and select similar completed projects based on profile indicators. Those new tools are working closely together with existing EPOS process tools to provide an integrated PSEE in which process evolution is effectively managed.

Validation

Finally, we describe a case study which has been conducted against a software company providing banking applications. The proposed categorization framework is applied to record and categorize process evolution collected from a sample of both completed and ongoing projects. Data has been collected by using a set of paper-based forms and interviews. The evolution and other useful baselines are then established and transferred to the EPOS experience database for analysis. Most critical and frequent evolution patterns are recognized, and empirical relations between typical evolution patterns and project profile are also identified. Results gained from the case study and their analytical data provide a quantitative foundation for better planning of new projects according to our proposed approach. In parallel, we also suggest some improvement initiatives in the future for the studied software company based on case study results and observations.
# Contents

1 Introduction ................................................. 1
   1.1 Background ............................................. 1
   1.2 Motivations ............................................ 4
   1.3 Research Strategy ..................................... 7
   1.4 Research Context ..................................... 8
   1.5 Summary of Contributions ............................. 9
   1.6 Structure of the Thesis ............................... 11

2 State-of-The-Art and Current Practices ..................... 15
   2.1 Introduction .......................................... 15
   2.2 Software Engineering ................................ 15
      2.2.1 Software Process ................................ 16
      2.2.2 Process-centered Software Engineering Environment - PSEE 17
         2.2.2.1 Process Programming - APPL/A ............... 17
         2.2.2.2 Net-oriented .................................. 17
         2.2.2.3 Rule-based .................................... 18
         2.2.2.4 Active database - ADELE .................... 18
         2.2.2.5 Role-oriented ................................ 19
         2.2.2.6 Work-flow - Process Weaver ............... 19
      2.2.3 Assessment of PSEEs ............................... 19
   2.3 Software Process Improvement Models .................. 20
      2.3.1 ISO 9000 ........................................... 21
      2.3.2 Capability Maturity Model (CMM) ............... 22
      2.3.3 Quality Improvement Paradigm (QIP) ............ 24
      2.3.4 BOOTSTRAP - An Assessment Method ............ 26
      2.3.5 Software Process Improvement and Capability Determination (SPICE) 28
      2.3.6 Discussion ........................................ 29
   2.4 Software Evolution .................................... 31
      2.4.1 What is Evolution? ............................... 31
      2.4.2 Evolution in Software Technology ............... 31
      2.4.3 Program Evolution Dynamics ..................... 31
         2.4.3.1 SPE classification ........................... 31
         2.4.3.2 Laws of Program Evolution ................... 34
         2.4.3.3 Feedback driven Evolution Process .......... 35
      2.4.4 PRISM - A Model of Change ...................... 36
      2.4.5 Discussion ........................................ 40
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.5</td>
<td>Software Measurement</td>
<td>40</td>
</tr>
<tr>
<td>2.5.1</td>
<td>Basic Measurement Theory</td>
<td>41</td>
</tr>
<tr>
<td>2.5.2</td>
<td>Measurement-based PSEEs</td>
<td>42</td>
</tr>
<tr>
<td>2.5.2.1</td>
<td>Ginger</td>
<td>42</td>
</tr>
<tr>
<td>2.5.2.2</td>
<td>Amadeus</td>
<td>42</td>
</tr>
<tr>
<td>2.5.2.3</td>
<td>ES-TAME</td>
<td>43</td>
</tr>
<tr>
<td>2.5.2.4</td>
<td>MVP-S</td>
<td>43</td>
</tr>
<tr>
<td>2.5.3</td>
<td>Discussion</td>
<td>43</td>
</tr>
<tr>
<td>2.6</td>
<td>Software Project Management</td>
<td>43</td>
</tr>
<tr>
<td>2.6.1</td>
<td>Software Cost Estimation</td>
<td>44</td>
</tr>
<tr>
<td>2.6.2</td>
<td>Scheduling and Resource Allocation</td>
<td>45</td>
</tr>
<tr>
<td>2.6.3</td>
<td>Project Monitoring and Review</td>
<td>46</td>
</tr>
<tr>
<td>2.6.4</td>
<td>Discussion</td>
<td>46</td>
</tr>
<tr>
<td>2.7</td>
<td>Examples of Software Experience Databases and their Usages</td>
<td>47</td>
</tr>
<tr>
<td>2.7.1</td>
<td>Experience Database at NASA-SEL</td>
<td>47</td>
</tr>
<tr>
<td>2.7.2</td>
<td>Experience data at Software Productivity Research (SPR)</td>
<td>48</td>
</tr>
<tr>
<td>2.7.3</td>
<td>Discussion</td>
<td>48</td>
</tr>
<tr>
<td>2.8</td>
<td>Evaluation and Open Issues</td>
<td>48</td>
</tr>
<tr>
<td>2.9</td>
<td>Chapter Summary</td>
<td>51</td>
</tr>
<tr>
<td>3</td>
<td>Objectives and Requirements</td>
<td>53</td>
</tr>
<tr>
<td>3.1</td>
<td>Introduction</td>
<td>53</td>
</tr>
<tr>
<td>3.2</td>
<td>Terminology Clarification</td>
<td>53</td>
</tr>
<tr>
<td>3.3</td>
<td>Objectives of Research on Software Process</td>
<td>56</td>
</tr>
<tr>
<td>3.3.1</td>
<td>General Research Objectives/Directions</td>
<td>56</td>
</tr>
<tr>
<td>3.3.2</td>
<td>Objectives of Process Support in PSEE</td>
<td>57</td>
</tr>
<tr>
<td>3.4</td>
<td>Our Underlying Research Objectives</td>
<td>58</td>
</tr>
<tr>
<td>3.5</td>
<td>General Research Issues</td>
<td>59</td>
</tr>
<tr>
<td>3.5.1</td>
<td>Overall Issues</td>
<td>59</td>
</tr>
<tr>
<td>3.5.2</td>
<td>Production Process</td>
<td>60</td>
</tr>
<tr>
<td>3.5.3</td>
<td>Meta-process</td>
<td>61</td>
</tr>
<tr>
<td>3.5.4</td>
<td>Process Support</td>
<td>61</td>
</tr>
<tr>
<td>3.6</td>
<td>Basic Requirements for a PSEE</td>
<td>62</td>
</tr>
<tr>
<td>3.6.1</td>
<td>Modeling Concepts</td>
<td>63</td>
</tr>
<tr>
<td>3.6.2</td>
<td>Process Modeling Language (PML)</td>
<td>63</td>
</tr>
<tr>
<td>3.6.3</td>
<td>Process Model Instantiation</td>
<td>64</td>
</tr>
<tr>
<td>3.6.4</td>
<td>Process Model Enactment</td>
<td>65</td>
</tr>
<tr>
<td>3.7</td>
<td>Additional Requirements Supporting Software Project</td>
<td>65</td>
</tr>
<tr>
<td>3.7.1</td>
<td>Data Management</td>
<td>66</td>
</tr>
<tr>
<td>3.7.2</td>
<td>Cooperation</td>
<td>66</td>
</tr>
<tr>
<td>3.7.3</td>
<td>Modeling Organizational Aspects</td>
<td>67</td>
</tr>
<tr>
<td>3.7.4</td>
<td>Supporting Project Management in a PSEE</td>
<td>68</td>
</tr>
<tr>
<td>3.8</td>
<td>Requirements for Managing Software Process Evolution</td>
<td>70</td>
</tr>
<tr>
<td>3.8.1</td>
<td>Evolution Classes</td>
<td>70</td>
</tr>
<tr>
<td>3.8.2</td>
<td>Bottom-up Process Model Evolution in PSEE</td>
<td>72</td>
</tr>
<tr>
<td>3.8.3</td>
<td>Unplanned Process Evolution in PSEE</td>
<td>72</td>
</tr>
<tr>
<td>3.9</td>
<td>Chapter Summary</td>
<td>73</td>
</tr>
</tbody>
</table>
4.6.2 Own contribution ......................................................... 114
4.6.3 Supervising work ....................................................... 115
4.7 Temporary Evaluation of EPOS baseline ......................... 115
4.7.1 Further Ideas for EPOS Extension ............................... 116
4.8 Chapter Summary .......................................................... 116

5 Modeling Experiences with EPOS ............................... 117
5.1 Introduction ............................................................... 117
5.2 Modeling a simple Waterfall Life Cycle ......................... 117
5.3 Modeling ISPW6 Example ............................................. 118
5.4 Modeling ISPW7 Example ............................................. 121
5.5 Modeling an Industrial Review Process Example ............... 124
5.6 Assessment of Gained Experiences ................................. 125
5.7 Chapter Summary .......................................................... 126

6 Categorization Framework for Evolution ....................... 129
6.1 Introduction ............................................................... 129
6.2 Basic Decisions .......................................................... 129
6.3 Categorization Framework for Process Evolution .............. 130
6.3.1 Basic Structure of the Framework ............................... 130
6.3.2 Framework Characteristics ....................................... 131
6.3.3 Constructing a Categorization Framework ................... 132
6.3.4 An Example of Constructing a Categorization Framework ...
   6.3.4.1 Short Background of the Studied Company ............... 133
   6.3.4.2 Identifying Questions in Dimension (step 1) .......... 133
   6.3.4.3 Elaborating the Aspects (step 2) ....................... 134
   6.3.4.4 Elaborating the Categories (step 3) ................... 136
6.4 Project Characterization Framework ............................ 136
6.4.1 Separation of Concern ............................................ 136
6.4.2 Project Profile Measures ......................................... 138
6.4.3 An Example of Project Characterization .................... 139
6.5 Relation between Process Evolution and Project Profile .... 140
6.6 Chapter Summary .......................................................... 141

7 Approaches to Manage Process Evolution .................... 143
7.1 Introduction ............................................................... 143
7.2 Basic Decisions .......................................................... 143
7.3 Overall Meta-process .................................................. 144
7.4 Evolution of Generic Process Model in EPOS ................. 145
7.5 Evolution of Enactable and Enacting Process Model in EPOS ...
   7.5.1 Manipulation of Task network Layout ..................... 147
   7.5.1.1 Task Insertion Operation ............................... 147
   7.5.1.2 Task Removal Operation ................................ 152
   7.5.2 Manipulation of Task network Scheduling ................ 154
   7.5.2.1 Revision of Task Schedule ........................... 154
   7.5.2.2 Revision of Task Resource ............................ 155
7.6 Evolution of Real-world Process .................................. 156
<table>
<thead>
<tr>
<th>CONTENTS</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.7 Empirical Evolution Planning Approach</td>
<td>157</td>
</tr>
<tr>
<td>7.7.1 Evolution Baseline Establishment</td>
<td>158</td>
</tr>
<tr>
<td>7.7.2 Initial Planning Support</td>
<td>159</td>
</tr>
<tr>
<td>7.7.3 Iterative Planning Support</td>
<td>160</td>
</tr>
<tr>
<td>7.7.4 Cost Estimation Technique</td>
<td>161</td>
</tr>
<tr>
<td>7.7.5 Evolution Pattern Analysis</td>
<td>162</td>
</tr>
<tr>
<td>7.7.5.1 Baseline Project Analysis</td>
<td>163</td>
</tr>
<tr>
<td>7.7.5.2 Inter-project Analysis</td>
<td>165</td>
</tr>
<tr>
<td>7.8 Tool Support</td>
<td>165</td>
</tr>
<tr>
<td>7.9 Approach Evaluation</td>
<td>165</td>
</tr>
<tr>
<td>7.10 Chapter Summary</td>
<td>166</td>
</tr>
<tr>
<td>8 Implementation Issues</td>
<td>169</td>
</tr>
<tr>
<td>8.1 Introduction</td>
<td>169</td>
</tr>
<tr>
<td>8.2 Requirements for Implementation</td>
<td>169</td>
</tr>
<tr>
<td>8.3 Experience Database in EPOS</td>
<td>170</td>
</tr>
<tr>
<td>8.3.1 Project Characterization</td>
<td>170</td>
</tr>
<tr>
<td>8.3.2 Process Evolution Categorization</td>
<td>173</td>
</tr>
<tr>
<td>8.4 Implementation Issues for Task Network Evolution</td>
<td>176</td>
</tr>
<tr>
<td>8.4.1 Manipulation of Task Network Layout</td>
<td>178</td>
</tr>
<tr>
<td>8.4.1.1 Task Insertion Operation</td>
<td>180</td>
</tr>
<tr>
<td>8.4.1.2 Task Removal Operation</td>
<td>181</td>
</tr>
<tr>
<td>8.4.2 Manipulation of Task Scheduling</td>
<td>181</td>
</tr>
<tr>
<td>8.4.2.1 Definition of Time Attribute to TaskEntity</td>
<td>181</td>
</tr>
<tr>
<td>8.4.2.2 Revision of Task Schedule</td>
<td>183</td>
</tr>
<tr>
<td>8.4.2.3 Revision of Task Resource</td>
<td>184</td>
</tr>
<tr>
<td>8.5 Tool to Support Empirical Evolution Planning</td>
<td>185</td>
</tr>
<tr>
<td>8.5.1 Project Selector</td>
<td>185</td>
</tr>
<tr>
<td>8.5.2 Evolution Analyzer</td>
<td>185</td>
</tr>
<tr>
<td>8.6 Integration of new tools into EPOS</td>
<td>187</td>
</tr>
<tr>
<td>8.7 Example of a Separate Experience Database at XXX</td>
<td>189</td>
</tr>
<tr>
<td>8.8 A WWW-based Experience Database Prototype</td>
<td>191</td>
</tr>
<tr>
<td>8.9 Chapter Summary</td>
<td>191</td>
</tr>
<tr>
<td>9 Case study</td>
<td>195</td>
</tr>
<tr>
<td>9.1 Introduction</td>
<td>195</td>
</tr>
<tr>
<td>9.2 The Experimental Paradigm</td>
<td>196</td>
</tr>
<tr>
<td>9.3 Case Study Context</td>
<td>196</td>
</tr>
<tr>
<td>9.3.1 Background</td>
<td>196</td>
</tr>
<tr>
<td>9.3.2 Quality System of XXX</td>
<td>197</td>
</tr>
<tr>
<td>9.3.3 Assessment of Current Practices at XXX</td>
<td>199</td>
</tr>
<tr>
<td>9.4 Case Study Design</td>
<td>201</td>
</tr>
<tr>
<td>9.4.1 Overall Objectives and Strategies</td>
<td>201</td>
</tr>
<tr>
<td>9.4.2 Research Question for the Case Study</td>
<td>203</td>
</tr>
<tr>
<td>9.4.3 Planned Activities for the Case Study</td>
<td>203</td>
</tr>
<tr>
<td>9.4.3.1 A1 - Preparation</td>
<td>204</td>
</tr>
<tr>
<td>9.4.3.2 A2 - Data Collection</td>
<td>205</td>
</tr>
</tbody>
</table>
9.4.3.3 A3 - Monitoring ........................................ 205
9.4.3.4 A4 - Tool Provision .................................... 206
9.4.3.5 A5 - Reuse of Experience Data .................... 207
9.4.3.6 A6 - EPOS Validation ................................ 207

9.5 Case Study Performance .................................... 208
  9.5.1 A1 - Preparation ...................................... 208
  9.5.2 A2 - Data Collection from Completed Projects ...... 214
  9.5.3 A3 - Monitoring Up-starting Project ............... 214
  9.5.4 A4 - Tool Provision ................................... 216
  9.5.5 A5 - Reuse of Experience Data ..................... 216
  9.5.6 A6 - EPOS Validation ................................ 217

9.6 Empirical Results ........................................ 217
  9.6.1 Project Profile ........................................ 217
  9.6.2 Evolution Status ....................................... 218
  9.6.3 Project Outcome ........................................ 219
    9.6.3.1 Estimation Capability ............................ 220
    9.6.3.2 Actual Effort Distribution ....................... 223
    9.6.3.3 Error Profile ...................................... 223

9.7 Evolution Analysis ....................................... 224
  9.7.1 Where-Why Frequency Analysis ....................... 226
  9.7.2 Where-How Frequency Analysis ....................... 226
  9.7.3 Typical Evolution Patterns .......................... 227
  9.7.4 Empirical Relations between Evolution and Project Profile ....... 228

9.8 EPOS Validation .......................................... 230
  9.8.1 Modeling XXX's Quality System ...................... 231
  9.8.2 Packaging Empirical Data ............................ 231
  9.8.3 Managing Typical Evolution Patterns in EPOS ...... 232

9.9 Improvement Suggestions .................................. 233
  9.9.1 Suggested Improvement Initiatives for XXX .......... 233
  9.9.2 Improvement Suggestions to EPOS ................... 234

9.10 Evaluation and Discussion ............................... 235
  9.10.1 Gained Benefits versus Objectives .................. 235
  9.10.2 Assessment on Stated Research Questions .......... 236
  9.10.3 Threats to Validity .................................. 237
  9.10.4 Lesson Learned ....................................... 237

9.11 Chapter Summary ......................................... 238

10 Evaluation, Conclusion and Further Work .................. 239
  10.1 Introduction ............................................ 239
  10.2 Summary of the Thesis .................................. 239
  10.3 Main Achievements ...................................... 240
  10.4 Evaluation .............................................. 241
    10.4.1 EPOS Highlights .................................... 241
    10.4.2 Evaluation wrt. Objectives ....................... 243
    10.4.3 Evaluation wrt. Basic Requirements for a PSEE ... 244
    10.4.4 Evaluation wrt. Requirement for Supporting Software Project .... 245
    10.4.5 Evaluation wrt. Requirement for Managing Software Evolution ... 245
10.5 Comparison with Existing Technologies ............................... 246
10.6 Criticisms to EPOS .................................................. 248
10.7 Further Work ......................................................... 249
10.8 Concluding Remark ................................................... 251

A  EPOS-PM Manual ......................................................... 253
   A.1 Introduction ....................................................... 253
   A.2 User Tutorial ...................................................... 253
      A.2.1 Hardware and Software Requirements .................. 253
      A.2.2 Installation and Starting up the system .............. 254
      A.2.3 Description of User Interfaces ......................... 254
         A.2.3.1 Prolog prompt ........................................ 255
         A.2.3.2 Agenda ............................................... 255
         A.2.3.3 Task Network ......................................... 256
         A.2.3.4 Mail tool ............................................ 258
      A.2.4 A simple process example ................................. 258
         A.2.4.1 Problem Description and Solution Proposal ....... 258
         A.2.4.2 Running through the example ....................... 259
         A.2.4.3 Single user environment - without delegation .... 260
         A.2.4.4 Multi user environment - with delegation ....... 261
      A.2.5 ISPW6 process example ..................................... 262
      A.2.6 Start working from the root_project ..................... 266
   A.3 Programmer Manual ................................................ 267
      A.3.1 Schema Manager ............................................. 267
      A.3.2 Schema Navigator ........................................... 268
         A.3.2.1 Navigating in Subtype Trees ....................... 269
         A.3.2.2 Navigating in Decomposition Tree ................ 269
         A.3.2.3 Manipulating Types Through Navigator .............. 270
      A.3.3 Editor ........................................................ 270
         A.3.3.1 Graphical Editor ..................................... 271
         A.3.3.2 Textual Editor ....................................... 272
      A.3.4 Modeling with SPELL ......................................... 272
         A.3.4.1 Argument Description ................................ 272
         A.3.4.2 SPELL Interpreter ..................................... 272
         A.3.4.3 Domain, Protection and Inheritance ............... 274
      A.3.5 Predefined Schema ........................................... 275
         A.3.5.1 Product ................................................ 277
         A.3.5.2 Activity ............................................... 278
         A.3.5.3 Relation ............................................... 278
      A.3.6 Modeling Methods ............................................. 279
         A.3.6.1 Instance-level ....................................... 279
         A.3.6.2 Type-level ............................................. 280
         A.3.6.3 Predefined predicates for defining task type .... 282
         A.3.6.4 Querying taskstate ................................... 284
         A.3.6.5 Creation of Product Structure ...................... 284
      A.3.7 Mailbox Facilities ............................................ 284
      A.3.8 Feedback ...................................................... 285
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>A.3.8.1</td>
<td>Predefined Types</td>
<td>285</td>
</tr>
<tr>
<td>A.3.8.2</td>
<td>Modeling with Feedback</td>
<td>285</td>
</tr>
<tr>
<td>A.3.8.3</td>
<td>Transmission Feedback at the same level</td>
<td>285</td>
</tr>
<tr>
<td>A.3.8.4</td>
<td>Transmission Feedback at different levels</td>
<td>286</td>
</tr>
<tr>
<td>A.3.8.5</td>
<td>Sending activities to the agenda</td>
<td>287</td>
</tr>
<tr>
<td>A.3.9</td>
<td>Dialog Box Library in XPCE</td>
<td>287</td>
</tr>
</tbody>
</table>
List of Figures

1.1 Research trend for software process ........................................... 6
1.2 Thesis structure and its inter-dependencies ................................. 14
2.1 External standards, a quality management system and quality plan .......... 21
2.2 Five levels in Capability Maturity Model ..................................... 23
2.3 QIP and Experience Factory .................................................. 25
2.4 The BOOTSTRAP key attributes .............................................. 28
2.5 The SPICE assessment framework ............................................. 29
2.6 S-type program [LB85] ..................................................... 32
2.7 P-type program [LB85] ..................................................... 33
2.8 E-type program [LB85] ..................................................... 34
2.9 The PRISM model of change ([Mad92]) ..................................... 37
2.10 Integration of CS-sheets with a DS-sheet ([Mad92]) ......................... 39
2.11 Example properties of a change ([Mad92]) .................................. 39
3.1 Software Process and their models in a PSEE ................................ 54
3.2 Software Evolution .................................................................. 71
4.1 EPOS overview ..................................................................... 76
4.2 An example of a Task Network ................................................ 77
4.3 EPOS Layer Architecture ...................................................... 79
4.4 EPOSDB mappings. .............................................................. 84
4.5 ER data model of EPOSDB predefined types ................................. 84
4.6 Transactions and their Ambitions and Choices ............................... 86
4.7 A SPELL example of type definition ......................................... 93
4.8 Predefined Types in EPOS-PM ................................................ 97
4.9 ER-schema for EPOS-PM ...................................................... 97
4.10 Predefined task states and State transition diagram ....................... 100
4.11 A Waterfall task type .......................................................... 101
4.12 A C-Compiler tool type ....................................................... 102
4.13 A C-File product type ........................................................ 102
4.14 A TaskDescriptor meta-type .................................................. 104
4.15 Schema Manager illustrating by a Navigator, Textual and Graphical Editor 107
4.16 Screen picture of Task Network Editor .................................... 109
4.17 Screen picture of CHAT ...................................................... 110
5.1 Necessary Types for Waterfall Example ...................................... 118
5.2 Example of simple waterfall life cycle in EPOS ............................. 119
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.3</td>
<td>ISPW6 Extended Types</td>
<td>120</td>
</tr>
<tr>
<td>5.4</td>
<td>ISPW6 task network</td>
<td>122</td>
</tr>
<tr>
<td>5.5</td>
<td>ISPW7 solution in EPOS</td>
<td>123</td>
</tr>
<tr>
<td>5.6</td>
<td>The cooperation transaction task network</td>
<td>123</td>
</tr>
<tr>
<td>5.7</td>
<td>The task network of Review process</td>
<td>126</td>
</tr>
<tr>
<td>6.1</td>
<td>Basic Structure of the Categorization Framework</td>
<td>131</td>
</tr>
<tr>
<td>6.2</td>
<td>A fully constructed categorization framework for process evolution</td>
<td>137</td>
</tr>
<tr>
<td>6.3</td>
<td>Hierarchy of project profile measures</td>
<td>139</td>
</tr>
<tr>
<td>7.1</td>
<td>Meta-activities to evolve generic process model in EPOS</td>
<td>145</td>
</tr>
<tr>
<td>7.2</td>
<td>Adding task T4 to the task network</td>
<td>148</td>
</tr>
<tr>
<td>7.3</td>
<td>A task instance and its AND-ports</td>
<td>150</td>
</tr>
<tr>
<td>7.4</td>
<td>Example of adding Review to the task network</td>
<td>151</td>
</tr>
<tr>
<td>7.5</td>
<td>Removing task T2 from the task network</td>
<td>152</td>
</tr>
<tr>
<td>7.6</td>
<td>Example of removing Review from the task network</td>
<td>153</td>
</tr>
<tr>
<td>7.7</td>
<td>Evolution based initial planning support</td>
<td>160</td>
</tr>
<tr>
<td>7.8</td>
<td>Evolution scenarios for analysis</td>
<td>164</td>
</tr>
<tr>
<td>8.1</td>
<td>New EPOS-PM types for project profile</td>
<td>170</td>
</tr>
<tr>
<td>8.2</td>
<td>Project Characterizer</td>
<td>172</td>
</tr>
<tr>
<td>8.3</td>
<td>New EPOS-PM types for evolution categorization</td>
<td>173</td>
</tr>
<tr>
<td>8.4</td>
<td>An example of a categorization tree</td>
<td>175</td>
</tr>
<tr>
<td>8.5</td>
<td>Evolution Registrar</td>
<td>177</td>
</tr>
<tr>
<td>8.6</td>
<td>Task insertion operation in EPOS</td>
<td>180</td>
</tr>
<tr>
<td>8.7</td>
<td>An example of prompted notification box for <code>cedit</code></td>
<td>181</td>
</tr>
<tr>
<td>8.8</td>
<td>Task removal operation in EPOS</td>
<td>182</td>
</tr>
<tr>
<td>8.9</td>
<td>Schedule and resource revision in EPOS</td>
<td>183</td>
</tr>
<tr>
<td>8.10</td>
<td>Two new task states and extended state transition diagram</td>
<td>184</td>
</tr>
<tr>
<td>8.11</td>
<td>Project Selector</td>
<td>186</td>
</tr>
<tr>
<td>8.12</td>
<td>Evolution Analyzer</td>
<td>188</td>
</tr>
<tr>
<td>8.13</td>
<td>Interaction between EPOS process tools</td>
<td>189</td>
</tr>
<tr>
<td>8.14</td>
<td>Evolution Category</td>
<td>192</td>
</tr>
<tr>
<td>8.15</td>
<td>Evolution Cost</td>
<td>192</td>
</tr>
<tr>
<td>8.16</td>
<td>Evolution Description</td>
<td>193</td>
</tr>
<tr>
<td>9.1</td>
<td>Project Model of XXX</td>
<td>198</td>
</tr>
<tr>
<td>9.2</td>
<td>System Development Model of XXX</td>
<td>200</td>
</tr>
<tr>
<td>9.3</td>
<td>Phasing of Activities to be performed in the Case Study</td>
<td>204</td>
</tr>
<tr>
<td>9.4</td>
<td>Project Characterization Form</td>
<td>211</td>
</tr>
<tr>
<td>9.5</td>
<td>Evolution Registration Form</td>
<td>212</td>
</tr>
<tr>
<td>9.6</td>
<td>Final Project Status Form</td>
<td>213</td>
</tr>
<tr>
<td>9.7</td>
<td>Error Report</td>
<td>215</td>
</tr>
<tr>
<td>9.8</td>
<td>Average percentage of Evolution Origin</td>
<td>219</td>
</tr>
<tr>
<td>9.9</td>
<td>Under- and over-estimation ECI associated to life-cycle activities</td>
<td>221</td>
</tr>
<tr>
<td>9.10</td>
<td>Actual Effort Distribution for XXX</td>
<td>224</td>
</tr>
<tr>
<td>9.11</td>
<td>Error Profile for project A and B</td>
<td>225</td>
</tr>
<tr>
<td>9.12</td>
<td>Case study at XXX and EPOS</td>
<td>230</td>
</tr>
</tbody>
</table>
## LIST OF FIGURES

9.13 Modeling selected parts of XXX’s process models in EPOS ........................................ 232
A.1 Agenda window .............................................................................................................. 255
A.2 Project Status Information Window ............................................................................... 257
A.3 Task Network window .................................................................................................... 257
A.4 Simple process example. ................................................................................................. 259
A.5 Specification of top-level task `modify_code` ............................................................... 260
A.6 Complete task network for `modify_code` process. ......................................................... 261
A.7 ISPW6: Start work ........................................................................................................... 263
A.8 ISPW6: Delegation .......................................................................................................... 263
A.9 ISPW6: Real task network with delegation. ................................................................. 264
A.10 ISPW6: Real task network without delegation. ......................................................... 265
A.11 Create composite project .............................................................................................. 266
A.12 Load Types .................................................................................................................. 267
A.13 Schema Manager illustrating by a Navigator, Textual and Graphical Editor ................. 269
A.14 The decomposition tree of task type `develop` ............................................................ 270
A.15 EPOS Predefined Types. ............................................................................................... 276
A.16 confirm_dialog ............................................................................................................... 287
A.17 view_dialog .................................................................................................................. 288
A.18 button_dialog ............................................................................................................... 288
A.19 editor_dialog ............................................................................................................... 289
A.20 text_dialog/4 ............................................................................................................... 289
A.21 multi_text_dialog ......................................................................................................... 289
A.22 text_dialog/5 ............................................................................................................... 290
A.23 info_dialog .................................................................................................................. 290
A.24 info_dialog including user-defined buttons ............................................................... 290
List of Tables

2.1 Comparison of five software process improvement models ................................... 30
4.1 Section Overview .................................................................................................. 80
6.1 An Example of Project Characterization .............................................................. 140
9.1 Project Profiles for five studied projects ......................................................... 218
9.2 Evolution Profiles for five studied projects ..................................................... 218
9.3 Percentage of Evolution Origin from five projects .......................................... 219
9.4 Estimation Capability ......................................................................................... 220
9.5 Estimation Capability Indices associated to life cycle activities ...................... 222
9.6 Actual Effort Distribution over life cycle activities for five projects ................. 223
9.7 Frequency of Evolution Occurrences distributed over Causes ....................... 226
9.8 Frequency of Evolution Occurrences distributed over Impacts ...................... 227
9.9 Evolution Patterns versus Approaches in EPOS ............................................. 233
Preface

Context of Work

This work has been done in the Norwegian University of Science and Technology (NTNU) from 1992-96. The thesis advisor has been Professor Reidar Conradi. I have been a member of EPOS project during this time and participated in implementing the EPOS prototype.

During my PhD study I have mainly been funded by a scholarship from REBOOT-drimg. This is a dr.ind education project associated with Norwegian REBOOT activities, and which is by itself funded by NFR under grant 101034/420. Simultaneously, I have also received support from the EPOS project, which was funded by NFR under grant ED0224.18457.

Acknowledgment

The following persons are acknowledged for contributions to the work:

- Professor Reidar Conradi, for inviting me to a PhD study, and for all help and guidance.

- All former as well as present members of the EPOS project: Letizia Jaccheri, Chunnnian Liu, Per H. Westby, Cristina Mazzi, Amund Aarstein, Weicheng Zhu, Luca Passani, Marianne Hagastøl, Bjørn Munch, Jens-Otto Larsen and Alf-Inge Wang. They are all acknowledged for cooperative work and many useful technical discussions.

- Other colleagues in the Software Engineering Group (SU-gruppe) at the Department of Computer and Information Science: Guttorm Sindre, Øystein Nytrø Sivert Sørmsgård, Eirik Tryggeseth, Geir Magne Høydalssvik, Sigurd Thunem, Terje Totland, Bjarte Østwold, Andreas Brendsten. They all are acknowledged for valuable general discussions and social contact.

- To all persons at the studied software company who have involved and actively helped me to conduct the case study. Especially, I would like to thank Tove Røsand and Hermann Bjugan for their belief and support in my work.

- To all anonymous referees of all conferences and workshops for constructive reviews and comments to my published papers.

- Finally, a special thank to my wife Linh for her enormous patience, understanding and support during this time.

xvii
Chapter 1

Introduction

1.1 Background

Over the last three decades software has gained increasing attention and constituted an enormous impact on almost everything we do. The software application is covering a wide domain ranging from space shuttle control system, airline reservation system, banking system, weather forecast to even simple control unit of housekeeping equipment. The influence of and the need for computer support will only increase in the future. Simultaneously, the requirements to the problem to be solved are also increasing in both complexity and size. The customer expectation and awareness of required software quality are growing with accelerating speed. This tendency of enormous demand has resulted in a need for developing software in such a corresponding speed during the last decades.

Software is characterized as a most troubling technology due to its labor-intensiveness and its required human creativity and skill. Nevertheless, the software development is the most error-prone activity because of its complexity in term of size, logic and computation. On the other hand, software developers are always put under pressure to produce highest quality deliverables as possible within a limited cost and time. Software development is therefore a highly risky business in which predictability is a key factor for success.

The term “software crisis” which was introduced at a conference in Garmisch-Partenkirchen 1968 [NR69], denoted the fact that the software costs were rising rapidly while the hardware cost were falling. Existing development technologies for small systems were unable to be scaled up to complex and large software systems. Moreover, the profit and business competitive pressures had primarily driven the business in many software suppliers. Their primary goals were thus reduction of time-to-market, increasing economical profits and gaining competitive advantages at the expense of software quality. Little efforts had thus been spent in defining and improving software development methodologies. Consequently, a large number of software projects were either failed or late in delivery, exceeded all budgeted cost and scheduled resources. Delivered products contained missing functionalities, severe defects and were thus costly to manage. It resulted in customer dissatisfaction, and forced software community to innovative thinking.

Software Engineering

The field of software engineering has therefore raised more and more interest among academic researchers and software practitioners who have realized a need for a scientific method for develop-
ing and maintaining high quality software within predictable cost and time frame. It therefore gave birth of a Software Engineering discipline. Software Engineering is concerned with software systems built by teams rather than individuals, uses engineering principles in the development of these systems and includes both technical and non-technical aspects [Som92]. Many software engineering methods and techniques (e.g., Yourdon’s functional method; Chen’s data-driven method; Rumbaugh’s object-oriented methods; etc …) have since then been developed according to different needs and application domains. New programming languages and supporting tools have also been developed and integrated into a single environment. Numerous equivalent terms have been proposed for such tool environments. The most common are Software Engineering Environment (SEE), Computer-aided Software Engineering (CASE) and Integrated Project Support Environment (IPSE). Such environments which are often identified as Product-centered Software Engineering Environments, include an integrated set of software tools to reduce the software engineering practices to the minimal usage of efforts and duration of time. Software systems are thus developed faster and easier by effectively exploiting supports offered by these CASE tools. Typical support includes software configuration management (SCM), version control and integrated programming facilities (editor, compiler, linker, debugger, …).

Software Process

The identification of “software crisis” has also led to the view that software development process is like other engineering processes, e.g., construction, manufacturing. The focus shift from product to process in which the product is developed and evolved, has drawn much attention in research community. The research on software process has been considered as a vehicle to improve software production. The term software process is used to denote all activities performed within an organization in order to develop software. The overall goal of studying and improving software process is to improve the software product. This goal stemmed from the claim that the quality of the software development process strongly influences the quality of the product being developed. Several process life cycle models (or generic process models) were thus introduced, e.g., waterfall, evolutionary, prototyping, spiral, transformational, etc ….. Those life cycle process models intend to describe software development and maintenance activities in a structured but coarse-grained manner. Thus, they are hard to be applied without customization and specialization. As software process turned to be more complex and involved a large amount of activities, generic life cycle models became short in satisfying a certain degree of details and control. An explosion of research on process management (PM) technology (or software process technology) has therefore been emerged and many Process-centered Software Engineering Environments (PSEEs) are consequently designed and implemented. The definition of process management (PM), as provided by the Software Engineering Institute (SEI), is “the application of process engineering concepts, principles, techniques and practices to explicitly monitor, control and improve the software development processes”. Software process technology, represented by those PSEEs, offers formalisms (process modeling language - PML), methods, mechanisms and tools to describe real-world software development process in term of models which are then configured, executed within a context of a project, and continuously evolved to adapt to new changes and needs. Typical support includes modeling, analysis, enactment and evolution. Generally, Software Process Technology mainly addressed technical aspects and not organizational and social aspects. The PSEEs mainly focus on process definition, execution, and evolution without seriously considering the support to configuration management and version control from SEE or CASE tools, and project supports (see below).
1.1. Background

Software Project Management

Software system is complex and its development is normally organized as projects involving several people across the line organization in software company. The high failure rate, late delivery of large projects has emerged increasing attention to software project management. People begin to realize that the problem stems as much from the management techniques used in carrying the projects as the engineering techniques to develop the system. Project management is therefore established as a research discipline which mainly addresses project process and support, such as planning, cost estimation, resource allocation, scheduling activities and monitoring progress. Such project supports can be seen as complementary to software process technology and PSEE to exploit mutual benefits of each others.

Software process improvement (SPI)

Despite of applying the best currently available methods, technologies and tools, we still don’t achieve the desired control and management of quality to delivered software systems. The requirement to product quality and thus to the way product is developed, becomes higher and higher over time. It has thus started a new research discipline whose main objective is to continuously improve software process. Currently, software process improvement (SPI) research activities and their applications are widely spread over different directions. SEI’s Capability Maturity Model (CMM) and ISO-9000 series represent one community. In this community software production process is assessed, evaluated against a well-recognized but rigid best practices in similar business application as an ordered, stepwise road-map for making improvement (a normative model). It is sufficient to demonstrate tangible evidences on achievement of prescribed process areas in order to be certified or ranged in a certain maturity level (also referred to as quality certifiers). However, there exists no specification of the technologies to be applied to achieve a particular process area. Another community of SPI models, represented by e.g., SPICE, BOOTSTRAP basically elaborates local and corporate-specific improvement plan based on the collected result of questionnaires and interviews. Those models are also categorized as quality certifiers, but differ from ISO-9000 and CMM in their customizable and flexible approach to define the improvement actions plan. The third SPI-community is represented by Quality Improvement Paradigm (QIP). QIP assesses existing process; states measurable improvement goal; selects appropriate methods or technologies that likely meet the improvement goals; installs and executes them; and analyzes/packages gained experiences for future use. QIP puts emphasis on the mainstream of project and quality management including project-level planning and metrics. Therefore QIP is highly oriented towards software practitioners. The last two communities require a deep insight into currently available practices which likely meet the identified improvement objectives.

Commonality for all SPI methods mentioned above is that they all by assessment recognize and correct existing inefficient processes by adopting most appropriate technologies. One of the problems of SPI initiatives is to obtain full management commitment since all SPI methods involve long-term and up-front financial investment. The management is hardly persuaded unless demonstrable benefits/gains of improvement investment are visible after a short period of time. Lacking of such evidences, the management can become a critical obstacle for improvement activity. On the other hand, people is by nature very susceptible to changes in their secure working environment unless significant and persuasive evidences are present. We are thus addressing the problem floating in the borderline with other disciplines involving human being, such as psychology, social science, etc. All those above have complicated the wide acceptance and practice of existing SPI methods. It has been illustrated by a limited
amount of improvement experiments based on SPI methods against software industry. Even if such experiments do exist, their gained results or success histories are not necessarily applicable for others.

Software Evolution

The Merriam-Webster's Collegiate Dictionary has following definitions for evolution: 1) a process of change in a certain direction; 2) a process of continuous change from a lower, simpler, or worse to a higher, more complex, or better state; 3) a process in which the whole universe is a progression of interrelated phenomena.

The world in which software systems are developed and operating, is of dynamic nature and continuously, rapidly evolving independently of used technologies and operating context. The released software systems are constantly subjected to changes due to i) detected defects after initial installation (corrective); ii) demand of adjustment for a new platform (adaptive); or iii) new customer needs or availability of new better technology (perfective). Software systems are thus continuously evolved during its life time (both development and maintenance phase). Evolution of software systems keeps occurring until it is considered as less cost effective than developing a new system or re-engineering the existing one. The evolutionary phenomenon of software systems has been studied thoroughly by Lehman over a long time.

The process in which such software systems are developed, also undergoes evolution. We want to distinguish between planned and unplanned process evolution. The former represents intended improvement initiatives (e.g. SPI methods mentioned above) towards an initially defined goal, e.g., to attain ISO-certification; to improve process maturity and capability by moving to higher CMM levels. On the other hand, the unplanned process evolution describes unanticipated changes, unexpected disturbances or distortions, and change pressures emerged during software projects. While the SPI is planned, implemented and fully aware of by software personnel, the unanticipated process evolution often imply critical impact on project schedule, progress and thus the desired quality of deliverable systems. The planned evolution\(^1\) has been extensively investigated for some time. As a result, several SPI methods have been elaborated (as described above), and many PSEEs have accordingly defined adequate technologies and architectural designs to effectively manage such evolution. On the contrary, unplanned evolution has not been focused on by the same extent.

1.2 Motivations

The historical background above shows how much research effort has been dedicated to software over long time. The ultimate research goal is to continually improve the software quality fully satisfying user requirements at any time by obtaining control over software processes. The history also reflects the divergence by which research has been directed in the last decades. Some main research directions or trends are identified in [Con96] as SPI categories.

Quality certifier The main focus is to improve organization's formal process standing by achieving certain recognitions like ISO-9000 certification, maturity levels in CMM or by following SPI-

\(^1\)Evolution hereafter means process evolution if something else are not specified explicitly.
1.2. Motivations

methods such as SPICE, Bootstrap. The underlying assumption is that rigorous process conformance will lead to better product and successful projects.

**Formalist** The main concern is to define a better process modeling language (PML) and highly characteristic for European SPI research which is represented by the first PROMOTER [FKN94]. The research tries to find a minimal set of core process elements to describe software process. The core requirements for PML is to facilitate understanding, communication and analysis. In addition, the formalist also strive to formally describe critical aspects in software development process such as Software Configuration Management (SCM), Cooperative Work, etc.

**Technologist** or tool advocate. The main research interest is to provide computer-assisted support for defining, configuring and executing software process models. Such research puts emphasis on assistance, guidance, enforcement, automation and monitoring in performing software process. New technologies such as WWW, CORBA attempt to be applied to full extent to develop PSEEes, workflow system, CSCW ranging from a single user to fully distributed and federated environments.

**Pragmatist** The research emphasis is directed towards software practitioners with focus on project and quality management, applying project-level planning and metrics, but with little formal process modeling (formalist) or enactment support (technologist). We can mention the work by NASA-SEL, resulting in the QIP including Experience Factory for reusing best practices, and Goal/Question/Metric approach for identifying suitable metrics. In addition, this research trend has resulted in a set of project management tools and features which are not satisfactorily provided by any PSEEes.

**Ideologist** The major concern is to extensively examine unmanaged software phenomena in order to lead to a scientific theory. We can mention the fundamental work by e.g. Lehman in studying software evolution, or the terminology work in [FH93, Lon93, CFP94].

**Organization reformer** The main goal is to make major innovative changes to the organization including policy, working routines, procedures and standards to better achieve the strategic goals. Work on enterprise modeling and Business Process Re-engineering is typical for this research direction.

The six categories are depicted in figure 1.1. The first four categories produce theoretical models and frameworks for SPI and are therefore referred as a *theory* group. Likewise, the technologist and pragmatist categories constitute a *system* and *practice* group. A more detailed description of those methods and systems can be found in chapter 2.

The above research trend gives us following motivations for our work in this thesis. Those motivations which are elaborated to concrete research objectives in section 3.4, will be evaluated in section 10.4.2 based on our work achievements.

**MI. Fully applying theory to system** The research trend illustrates a full coverage of theoretical models and methods for SPI. As the chapter 2 will present later, research on software process technology represented by a large number of PSEEes has been initiated to provide computerized support for those theories. Still, the PSEEes have poorly exploited or built upon all the theoretical SPI work. The development of PSEEes has occurred independently of and divergently to the SPI underlying needs and priorities. This has been demonstrated by the fact that most PSEEes put
stronger emphasis on formal process modeling language and enactment support of process models than e.g. product version/configuration management (SCM), project management, process model evolution and SPI by experience learning. Such narrow research focus of PSEE which has contributed to an increasing of process awareness, implies a marginal impact on practical value of software process improvement.

The quality certification models are very general and lack immediate usability. Further, the models are not geared towards small and medium sized development teams or companies which are a typical dimension of software companies in Norway. It is therefore necessary to down-scale the SPI models by extracting relevant aspects to be adapted to small cases. E.g. the underlying thought of SPI models regarding to process assessment (SPICE, Bootstrap) to identify critical process areas and experience learning (QIP) is highly relevant for conducting process improvement within a small format.

M2. Investigating the process evolution issue  The world in which software systems are developed and operated, is of dynamic nature. It thus results in a series of changes, evolutions or disturbances to both software systems and its development process. The study of software system evolution has been extensively done by Lehman, but the study of process evolution has not been paid by the same interest. Process evolution is complex by nature. It is thus impossible to precisely define what it is constituted of and therefore to plan in advance its impact to software project. It has also claimed that such evolution is both hard and costly to manage. The fact that process evolution is unique and encompassing different aspects makes it hard to understand without a systematic categorization. Based on a fully comprehensive understanding of the process evolution we can find a way to manage it by exploiting the existing methods or approaches.
1.3. Research Strategy

M3. Bridging the theory/practice gap The ultimate goal for process research is to achieve demonstrable gains of SPI models to software practitioners. By doing that, the SPI culture will have a chance to be institutionalized in the software organizations. While there are a large number of theoretical models for SPI, their validation in real industrial setting is still scarce. Therefore, process community has gained far too little experience in the adaptability and potential gain of such methods in actual processes. Such a theory/practice gap can be bridged by using software process technology (e.g. PSEE) as a means. Researchers should provide practitioners new or revised methods and hypotheses, possibly with some languages and instrumented tools. These hypotheses should be tried out in an industrial setting, either real or simulated. Inversely, industry should return real-world and metricated process models and experiences with these. This feedback can be analyzed in a research situation, and provide ground for revised SPI methods, and so on. Such a mutual and long-term cooperation on SPI issues is often lacking.

1.3 Research Strategy

In order to direct our research focus to satisfy the identified issues above, we have chosen following suitable strategies to assure effective achievement.

Survey and analysis The relevant software practices are investigated based on published material from different journals, conference/workshop proceedings. It is important to gain a full and correct insight into the current practices concerning software process technology (SPT), software process improvement (SPI) and related research fields in the last decades. Based on this insight, work objectives and requirements can be stated. Such survey and analysis are presented in chapters 2 and 3 respectively.

Theory elaboration An instrumentable categorization framework is defined to better understand the properties as well as the consequences of process evolution. Appropriate approaches are designed for a PSEE to effectively manage different kinds of process evolution both before and during project execution. The theoretical work is presented in chapters 6 and 7.

Design and implementation A PSEE prototype is first designed and implemented to satisfy the basic requirements. The tool support for the theoretical work on managing process evolution is then integrated into the same PSEE prototype. The development and extension of the PSEE prototype are described in chapters 4 and 8 respectively.

Validation The PSEE prototype is first validated with well-known process examples and then with a full-scale industrial setting as a case study. The validation and the case study are chapters 5 and 9 respectively.

Note that there is no single, universally applicable silver-bullet remedying all the dilemma. A large number of methods and techniques has been elaborated with different focus and objectives. It is therefore important for us to prevent from discovering the wheel again. We should rather provide a full insight into, learn from the best practices and direct our research to the right track. An appropriate support and guidance is more flexible than straight-jacket or strict formalism.
1.4 Research Context

Our research activities have essentially conducted within following contexts:

**EPOS** stands for Expert System for Program and (Og in Norwegian) System development. This project had been supported by the Royal Norwegian Council for Scientific and Industrial Research (NTNF) through grant ED0224.18457. The EPOS project was from 1986-1990 a joint R&D project between our Software Engineering group and ELAB-UNIT in Trondheim, and Veritas Research and Sysdeco on Oslo. The initial and primary project goal was to implement a software engineering database exploiting participants’ expertise on Object-oriented (OO) technology, OO database, and configuration management (CM). It resulted in a prototype of EPOSDB. After that, the EPOS project has been partly funded by Norwegian Research Council (NFR) through the REBOOT project. The process management (EPOS-PM) began first then to be focused in the project. The entire project relies on resources from funded PhD students, and under-graduated students through various projects. The first EPOS prototype was initiated in 1989 with a main focus on software configuration management (SCM). The author joined the EPOS group since 1992, and has been one of the main contributors for researching and developing the process management (PM) part of EPOS as well as the work on integrating SCM and PM in EPOS.

**PROMOTER** was the EU project #7082 in 1992-1995, an ESPRIT basic research action working group (BRA WG) on software process modeling. The Norwegian research council financed the Norwegian participation, covering marginal costs of travel and office expenses. The Norwegian partner was NTH, with Prof. Reidar Conradi as project responsible. The other participants were Univ. Nancy (coord.), Univ. Grenoble, Univ. Manchester, Imperial College, Univ. Dortmund, Univ. Leiden, Politecnico di Milano, and Univ. Pisa. The PROMOTER goal is to consolidate and enhance European research in Software Process Technology. The project has provided a fruitful context for discussion and research exchange. The project has issued one book on Process Modeling [FKN94]. The author participated in writing a chapter about EPOS. Another book is scheduled for 1996, titled “Software Process Technology”. The author participated in the group working with editing the chapter on meta-process in the period from 1993 to 1994 together with Univ. Manchester and Univ. Grenoble. This work has provided valuable insights and motivations about process evolution for the research in this thesis.

**SPIQ** Software Process Improvement for better Quality project is sought to be executed in 1996-2001 by four research and 10-12 software producing organizations. The latter are mainly small and medium enterprises. They operate in different domains, apply different development technologies, and represent different sizes (teams of 2-10 persons) and company cultures. The SPI emphasis will be on different development phases (planning vs. requirements vs. testing), and be driven by different needs (time, budget, reliability etc.). This heterogeneity is both a problem and an advantage. The industrial goal is to realize a 30% SPI saving in costs in these companies within 5 years, and to convert these companies into learning organizations. The research goal is to produce a Norwegian SPI method book by down-scaling, adapting and enhancing existing SPI methods. An experience database will be exploited, possibly augmented by commercial workflow/PSEE tools. SPIQ gives the author an opportunity to get in contact with one of the industrial partner which is willing to cooperate in validating the categorization framework for process evolution to improve estimate accuracy.
1.5 Summary of Contributions

The author contribution is reflected in chapter 5 through chapter 9, plus appendix A. The chapter 4 is mostly a result of cooperative effort, with the author as one of main contributors. In the following, we summarize the contributions and annotate with the international publications (i.e. refereed conference papers, or book articles) which the author has been involved.

Initial Implementation of EPOS

The author has been one of the main contributors to EPOS prototype from 1992-1996 [CHL+94a, MCL+95]. The claimed contribution is presented in details in section 4.6. The author has jointly designed and implemented EPOS-PM’s process modeling language SPELL [CJM+92, NC94c] and kernel process models including meta-process model [NC94b].

The author has designed and implemented support for managing cooperative work within the context of EPOS-PM by introducing flexible cooperation protocol and associated workspace layout [NC93, Ngu93]. The author has also introduced high-level language macros in SPELL to improve the clarity of the modeling language. An integrated project manager tool is initially designed and implemented to provide support for project initiation, delegation, simple tracking and effort collection. The author has designed and implemented the communication facilities and feedback mechanism between activities in EPOS-PM. In addition, the author has had a main responsibility to maintain all EPOS process tools over those years to incorporate new design features.

Apart from the described collaborative and own contribution above, the author has also supervised several students to implement the Schema Manager, Broadcast Message Server and Task Network Editor for EPOS. These tools are then maintained by the author to adapt to new design changes and needs.

Modeling process examples

The author has participated in designing the solution for process example ISPW7 in EPOS [CAJ+91]. The solution designs for both ISPW6 and ISPW7 in EPOS are re-designed and actually implemented in EPOS by the author. The reason for re-designing earlier solutions is to incorporate new EPOS features to better satisfy the example requirements. In addition, the author has modeled an industrial process for review in EPOS. All those modeling experiences and author contribution are found in chapter 5. The modeling experiences have been useful to validate our work and motivate for further research.

Categorization framework for process evolution

The author proposes a flexible framework to categorize process evolution during project [NC94a, NC96]. The framework is based on three abstraction levels: dimension, aspect and category. The dimensions specifies pertinent questions like: where evolution is originated; why evolution is caused; what entities are affected; when evolution is detected/corrected; how evolution is handled/reacted; and by-whom evolution is corrected. The framework aspect and category are then specialized to adapt to a particular organization according to a proposed construction algorithm. Collected evolution occurrences are recorded, structured, categorized and then associated to a project context in which they are captured.
A characterization scheme to classify projects into *profile, estimate* and *actual outcome* properties is therefore suggested. The project profiles measures or indicators which are associated with corporate-specific weights, are used to determine similarities between projects.

**Approaches to manage process evolution in EPOS**

The author elaborates different approaches to manage process evolution to different process model variations. A classification of process evolution is first introduced to direct the research focus. A *top-down* approach to address evolution of generic process models (i.e. types) in EPOS is then presented (also from [NC94b, NC94a]). *Bottom-up* approaches to manage evolution of enactable and enacting process models (i.e. task network instances or project plans) in EPOS are then described. They offer support to manipulate task network layout and revise task schedule and resource both before and during execution time. Such modifications which do not necessarily affect the corresponding generic models, are locally encapsulated and effective only during the project life time. At project completion, the modifications are evaluated and eventually incorporated to the corresponding generic process models. The case study in chapter 9 clearly indicates that the arrival rate or change pressures of bottom-up process evolution is much more frequent and has therefore greater impact on project outcome than of the top-down evolution.

An approach which is called *empirical evolution planning* is presented to support both initial and iterative project planning. The approach is mainly based on the project experiences or baselines, including e.g. estimation capability, evolution data that are recorded and categorized by the framework above. Initial estimates of new project can be improved by anticipating frequent evolution patterns and their associated impacts from either all completed projects or a single best-matching project. During project execution, estimates are also revised based on the recorded project experiences to interactively react to unanticipated circumstances. The approach requires an existence of an experience database which records a project history for each completed project. The project history includes profile characteristics, actual effort, estimation capability, evolution occurrences, and recognized evolution patterns. The proposed approaches are presented in an article which has been submitted to European Software Engineering Conference (ESEC) in 1997.

**EPOS extension**

The proposed categorization framework and approaches are incorporated in EPOS system by extending the existing data structure to represent evolution data and other relevant experiences. An experience database in EPOS is thus designed and implemented together with adequate tools to characterize project (*Project Characterizer*), record/categorize evolution occurrences (*Evolution Registrar*), analyze collected evolution data (*Evolution Analyzer*), and select similar completed projects (*Project Selector*). The new tools are working tightly together with the currently available process tools in EPOS. The existing Project Manager Tool is upgraded to provide dynamic re-planning supports (i.e. manipulation of task network layout and schedule/resource) for better managing evolution of enactable and enacting process models before and during project execution.
1.6 Structure of the Thesis

Empirical case study

A case study is conducted against a software company which provides banking applications. The objectives of the case study are among others to establish baselines for the studied organization regarding to unplanned evolution during projects, and then improve its estimation capability. The proposed categorization framework and approaches are thus applied to characterize and collect relevant data from 4 completed and 1 ongoing projects to build the empirical baselines. The data collection has resulted in 4 paper-based forms which will be institutionalized into the process of the studied company in the future. The established baselines include actual effort distributed over life cycle activities; estimation capability indices; frequency and impact of evolution occurrences; and typical evolution patterns. All such information is associated to a set of configurable project profile measures such as: project type, duration, staff size, staff skill and competence and customer information. The collected data are exported to and packaged in EPOS experience database. The analytical results are thus derived by applying newly implemented tools in EPOS.

The established baselines, although not statistically significant due to small project sample, schedules intend to be used in planning new projects. Unfortunately, there is no time to carry out this activity. Instead, we have suggested some improvement initiatives for the studied company in the future.

The empirical results together with analytical data from this case study are reported in [NWC97].

1.6 Structure of the Thesis

The structure of thesis is as follows:

Chapter 1: This introduction.

Chapter 2: State-of-the-art and current practices. We survey existing literature and focus on six relevant areas of topic: i) software engineering, software process and PSEE; ii) software process improvement models (e.g. ISO 9000, SEI’s CMM, QIP, BOOTSTRAP, SPICE) and their comparison; iii) software evolution, including Lehman’s program evolution dynamics and Madhavji’s PRISM model of change; iv) software measurement theory and investigation of measurement-based PSEE; v) software project management and its impact on process management and vice versa; vi) existing software experience databases together with their usage and benefits. An assessment of existing technology and an outline of further work directions is also included in this chapter.

Chapter 3: Objectives and Requirements. The chapter starts with a clarification of terminologies which will be used in the thesis. A set of research objectives is identified and discussed - starting from a high level of abstraction and moving towards specific objectives of process support in a PSEE. The work directions identified in chapter 2 are further elaborated to basic requirements for a PSEE; additional requirement supporting project management and software evolution. The stated requirements are used to evaluate our work in the rest of the thesis.

Chapter 4: EPOS baseline. The chapter describes the EPOS system as a PSEE prototype. The detailed descriptions of its architectural aspects and features are included. It is divided into two main layers: 1) version and configuration management (EPOSDB) and 2) process management
(EPOS-PM). A temporary evaluation of EPOS baseline regarding to the stated requirements in chapter 3 is provided.

Chapter 5: **Modeling experiences with EPOS.** The chapter presents result as well as gained experiences with modeling processes in EPOS. The process examples being modeled are: a waterfall life cycle model; ISPW6; ISPW7; and an industrial review process. The gained modeling experiences have provided us a basis for continual work to evaluate and then improve EPOS system.

Chapter 6: **Categorization framework for evolution.** The chapter presents our proposed categorization framework for process evolution and an algorithm to construct a detailed categorization framework for a particular organization. The chapter also proposes a framework how a project is characterized, and illustrates with a concrete example. A discussion on empirical relations between project profile indicators and process evolution is also presented.

Chapter 7: **Approaches to manage process evolution.** This chapter presents approaches to evolve process models in EPOS. Process models exist in three generic, enactable and enacting variants. An approach with associated tool support is used to evolve the generic process models in EPOS. Approaches which are inspired by re-plan operations in project management discipline, such as activity addition/removal and schedule/resource revision, are defined to evolve enactable and enacting process models in EPOS. An approach to support both initial and iterative planning based on **empirical evolution** data is elaborated based on the categorization framework in chapter 6.

Chapter 8: **Implementation issues.** The chapter presents in details how the proposed framework and approaches are realized in EPOS by both extending the existing data structure and providing adequate tool support. The experience database and newly implemented features in EPOS are then used to package and analyze empirical data from the case study (in chapter 9). The chapter also describes two experience database prototypes which are adopted our framework to record and categorize evolution data. One of them is implemented at the studied software company XXX. The another is built and makes available in the World Wide Web (WWW).

Chapter 9: **Case study.** The chapter presents a case study which is conducted against a software company providing banking applications. The proposed framework and the approach are used to characterize project, collect relevant data, and establish useful baselines to improve effort estimation of the studied software company. The collected data from a sample of five projects are fed back to EPOS for analysis. The most frequent evolution patterns are then identified, and improvement or corrective initiatives are suggested. Part of the development processes at the studied organization is modeled in EPOS to validate the new re-planning features in dealing with bottom-up process evolution.

Chapter 10: **Evaluation, conclusion and further work.** The chapter presents a summary of the thesis followed by an outline of our main achievements. We then evaluates the achievements against the requirement stated in chapter 3 and compare them to existing, related works. A conclusion and plan for further work are finally discussed.

Appendix A: **EPOS-PM manual.** The appendix encloses a manual for EPOS-PM which was written by the author to cope with the lack of documentation for both new and existing group members. The manual consists of two parts. The first part is a tutorial guide that makes the users familiar with EPOS-PM by running some existing process examples. The second part is a reference manual for process modelers to use EPOS process tools to describe and evolve process models.
Since the manual was written about two years ago, it certainly suffers from the lack of recently implemented features which are presented in chapter 4. However, it is included in the thesis for the sake of completeness and totality.

The thesis themes and their dependencies are graphically presented in figure 1.2. This figure provides the reader an overview of the implications of the topics in the thesis as well as the way they logically tight to each other. E.g. the graph conveys the ground for identifying design philosophy by means of motivations and objectives for our research, and the assessment of them against our achievements.
Figure 1.2: Thesis structure and its inter-dependencies
Chapter 2

State-of-The-Art and Current Practices

2.1 Introduction

This chapter will present the state-of-the-art in software process and encompass following six main topics. i) Software engineering and its process in section 2.2; ii) Software process improvement models in section 2.3; iii) Software evolution in section 2.4; iv) Software measurement in section 2.5; v) Software project management in section 2.6; and vi) Benefits of software project experience database in section 2.7. An evaluation is presented after the presentation of each topic. In section 2.8, we sum up our survey of existing technologies and outline further work directions which provide a foundation for the remainder of the thesis.

2.2 Software Engineering

The term software engineering is first introduced in the late 1960s at a conference in Garmisch-Partenkirchen [NR69]. The definition of software engineering in [Som92] is: “Software engineering is an engineering discipline concerned with the practical problems of developing large software systems. It is not just programming nor is it computer science. Software engineers must be professionals who use theory from other disciplines and apply this cost-effectively to solve difficult problems.” Software engineering is concerned with software systems built by teams rather than individuals, uses engineering principles in the development of these systems and includes both technical and non-technical aspects.

At the same conference in Garmisch-Partenkirchen the term software crisis was coined to describe the fact that software costs were rising rapidly while the hardware costs were falling. Existing development methods were not good enough for large and complex systems. That is, techniques which were applicable for small systems, could not be scaled up to handle increasing software complexity. Many software projects were failed or delivered with exceeding costs and schedule. Developed software was unreliable, difficult to maintain, lacking of required features. The software crisis has not been resolved yet although there have been great improvements in software engineering methods and techniques, in tools for system development and in the skills of development staff. The demand for software is increasing faster than improvements in software productivity. There is not a single solution to the problem of software engineering (“no silver bullet”) [Som92].

We need even better tools, techniques and methods and, perhaps most importantly, better education and training. This has given birth to a increasing research on Software Engineering Environments (SEEs).
SEEs are software aiming at providing computer based support for software engineering. The major objective of such tools is to provide comprehensive support in an integrated manner. Due to the integrating embedded facility the term environment was then introduced. According to the IEEE Standard Glossary a software engineering environment is:

"An integrated collection of tools accessed via a single command language to provide support capabilities throughout the software life-cycle. The environment includes tools for design, editing, compiling, loading, testing, configuration management and project management."

SEEs are also known as other names, e.g. Programming Environment (PE), Programming Support Environment (PSE), Integrated Product Support Environment (IPSE), Software Development Environment (SDE), Computer Aided Software Engineering (CASE).

Software Engineering is a very large topic. We have therefore chosen to focus on aspects which are considered as important from a process perspective. In the following sections, we will first present some typical software processes (i.e. life cycle models) together with their advantages and disadvantages. We then discuss Process-centered Software Engineering Environment (PSEE) in the next section.

2.2.1 Software Process

As the notion of software crisis is identified in the late 1960s, research on software development process began to receive increasing attention. It started with a waterfall life cycle model [Roy70] being made up by a number of stages such as requirements specification, software design, implementation, testing and so on. After each stage, there is a "signed-off" before the following step proceeds. After the proposal of the waterfall life cycle model, there has been an explosion of several models. They are e.g. transformational approach by Balzer [BCG83], evolutionary model by Basili [BT75], LST model by Lehman [LST84], spiral model by Boehm [Boe86], and prototyping approach [Dav92].

Balzer's transformational approach is an automatic paradigm to convert the formal specification of a software system into a program that satisfies the specification. Basili's evolutionary model is aiming at reducing the product delivery time of the waterfall life cycle model. By frequent product delivery cycles, one focuses on small incremental develop-deliver-measure-adjust steps. Lehman's LST model is another transformational approach to represent software process as a multi-step sequence. Each step involves base and target representations that may also be viewed as specification and implementation. Boehm's spiral model is a risk-driven approach. Its key characteristic is an assessment of management risk items at regular stages in the project and the initiation of actions to counteract these risks. A typical cycle in the spiral proceeds through iterative sequencing of steps leading to risk analysis and resolution for a given portion of the software product. The prototyping approach involves developing a program for user experiment. Its objective is to establish the system requirements followed by a re-implementation of the software to produce a satisfactory system.

The process life cycles above help us to become aware of and gain valuable insight into the software process and to determine the order of global activities observed in the production of software. These life cycles, especially waterfall model, are of great value from the management viewpoint. Due to the inherently intangible of software systems, the management hardly looks at the work itself to assess
2.2. Software Engineering

progress. By adopting the life cycles, management can measure progress by having visible artifacts (e.g. documents, reports) produced, reviewed and approved.

In general, they define general philosophy of sequencing high-level project activities, as well as customer validation of intermediate products. But, a life cycle hides important process details that are crucial for the success of software projects [Mad91], e.g. triggering and terminating conditions of an activity; state of a product component; inputs/outputs of an activity; data flow between activities; the human roles of process; the constraints and execution order of process steps. The life cycle models are thus designed to be general for a wide range of organization, domains, and situations. As a consequence, they do not usually fit any particular situation, and they are vague on the operational aspects, i.e. what to do in a given situation.

Recognizing such disadvantages of general life cycles models, we need to better understand the software process to be able to describe them. It is done by first creating formalisms suitable for describing and representing processes; by then automating the software process; and by developing methodologies, tools and environments for engineering and controlling the software processes. As we see below, several PSEE's try to address those issues. Software process model is therefore more fine-grained, specific and operational than life cycle or software engineering methods.

2.2.2 Process-centered Software Engineering Environment - PSEE

A PSEE is defined in [Lon93] as an environment which provides a direct support to its users by interpreting explicit automation-oriented software process models. The generic process interpreter or process engine is the heart of a PSEE. Existing PSEE's can be classified by the underlying paradigms of their process formalism or process modeling language (PML). Note that some PSEE's can as well be classified in several categories.

2.2.2.1 Process Programming - APPL/A

It is initiated by the famous phrase of Leon Osterweil: “Software Process are Software Too” in [Ost87]. This article has opened for possibility in exploiting well-known technologies in software programming to software process. A representative formalism for this way of thinking is APPL/A [JHO90].

APPL/A extends Ada with persistent relations, triggers, predicates, and transaction statements. A process model corresponds to an APPL/A program that is statically compiled into an executable program and then executed. Neither APPL/A programs, nor their executables may be manipulated and changed on the fly. However, relations are explicitly stored in the underlying storage management and together with triggers and transactions mechanisms are useful to manage product changes.

2.2.2.2 Net-oriented

SPADE [BFGL94] has been developed at Politecnico di Milano since 1990. It uses generalized and reflective Petri Nets expressed in the SLANG PML. These nets can be dynamically built and enacted by a Process Engine. Enaction usually means calling external tools via a BMS (DECFUSE). Inversely: the external tools are instrumented to send messages back via the BMS and a SPADE-filter which convert the message to a Petri Net token. There are no constraints on the
number of BMSs for one user, and the number of process engines depends on the active copies being enacted. The process model (Petri Net and its types) is stored in the O2 OO DBMS, while product files used by tools will be extracted out from the DB and stored in external workspaces. Activity definitions, type definitions and active copy (enacting variation of process model) are modeled both as code for execution and as datum for modification. Several strategies for propagation of process change may be applied.

MELMAC [DG90], generic process models are expressed as refinable FUNSOFT nets. A FUNSOFT net mainly consists of a hierarchical Petri net where each transition may be marked with a modification point denoting that the transition may be refined before executing it. A process model is instantiated by linking the transition of a FUNSOFT net to tools and where an initial marking is defined. The system does not exhibits reflection, because FUNSOFT net definitions are not explicitly represented nor modifiable by the system. Thus FUNSOFT net enables refinement or specialization, but not evolution by corrections.

2.2.2.3 Rule-based

MARVEL [BK91] a process model is specified by a rule set denoting the activity part, a project type set denoting the product part, and tool envelopes denote the tools. All these may be loaded into the kernel to obtain a Specific Process Model. Process evolution is supported by imposing a set of constraints that define the legal evolution steps. An evolution step is a change either to the definition of a single class or a single rule. Neither enactable process models nor executing software processes are explicitly represented. This implies that they cannot be inspected, recorded, or modified.

MERLIN [JPSW94] is a software process modeling language, where a rule based technique is used to build a knowledge base describing the software process. Rules and facts in the knowledge base may be interpreted in two forms: backwards and forward chaining. Backwards rules select the roles and the activities a user may perform. Forward chaining, instead, is applied when explicit guidance is provided by the system. MERLIN supports the basic abstractions of any software process, such as activities, software objects, roles and resources. Since rules and facts may be dynamically inserted and deleted from the knowledge base, the model exhibits great flexibility in representing changes that occur during process execution.

2.2.2.4 Active database - ADELE

ADELE [BEM94] has been developed at LGI at University of Grenoble since 1982. ADELE2 is primarily a software engineering database with process extensions. It is sold as a commercial product by Verilog, and is in daily use by over a dozen major software developers in Europe. ADELE is composed of two major components: the ADELE kernel covering low level object storage and coordination, and the ADELE-Tempo extension covering process management aspects. ADELE2 is primarily a reactive, versioned, client-server DBMS offering a DDL and DML, with a dynamically bound schema, nested short transaction and work space support. Tempo is the ADELE’s process extension. It defines a process model based on the concepts of role and connection. A role allows one to redefine dynamically the static and behavior properties of objects where playing that role within a process. A connection
expresses how processes collaborate. The rational of introducing role w.r.t. process evolution is that an instance can be dynamically bound to an arbitrary role at any time.

2.2.2.5 Role-oriented

**IPSE** [Sno92] enables to express process models in term of *Roles and Interactions* that are described by object-oriented classes. Executing roles may be evolved by introducing new classes of roles and interactions, creating new role instances, modifying the behavior of existing roles, and changing the network topology of the interacting instances. The IPSE PML is highly reflective: definition, customization, instantiation, execution, monitoring and evolution of a given process model are supported and well documented.

**Process Wise Integrator (PWI)** [BGR+94] has been developed since 1986 as part in the IPSE 2.5 project in the UK, the University of Manchester being a project partner. It is now hardened into a commercial product by ICL. It uses a role - activity - interaction paradigm to describe organization processes in an process programming language PML (Process Management Language) with a strong focus on evolution. The features of PML and its underlying language are inheritance, callable compiler and persistence. There is no need for the system to differentiate between contributions to the process by human users and by tools. Both can be represented by some combination of roles. Its emphasis is on flexible definition and enactment of “roles”, being typed and networked tasks allocated to certain persons and tools. PWI is reflective and can start new sub-activities on-the-fly, and distribution is well supported (even home “hook-up”).

2.2.2.6 Work-flow - Process Weaver

Process Weaver [Fer93] has been developed since 1988, and is now a commercial product from Cap Gemini Innovation in Grenoble. Its emphasis is on flexible support of simple task flow for decentralized process performers, and is not limited to software processes. Its process model is extended Petri nets, being interpreted at runtime and with tool invocations through a BMS. A network editor is used to make new Petri net fragments which can be dynamically incorporated (dynamic evolution). The emphasis is on activity modeling with agendas presented to the process performers, not on product modeling, although an integration with ADELE2 is under way. The process modeling focuses on three perspectives: project management (method level); team-interworking (cooperative procedure), and individual team member (work context).

2.2.3 Assessment of PSEEs

In [Lon94], Lonchamp has conducted an assessment of PSEE, including also those above, in the context of PROMOTER Working Group [FKN94]. The objective of the assessment is to highlight both the commonalities as well as specificities in capturing/describing process knowledge and guiding process performance. The assessment was conducted by defining a set of open questions which were answered by the designers of the PSEE themselves. The assessment result reveals that most PSEE are mainly oriented towards the guidance of process agents through the mechanical interpretation of process models. Few PSEE are primarily oriented towards human interpretation and/or mechanical interpretation
for analysis or simulation of processes. Most of PSEEs include, to a large extent, the relevant modeling concepts such as activity, artifact/product, agent, role, tool and constraints. Such PSEEs therefore offer an explicit representation of software processes in one or another formalism. E.g. APPL/A, MEL-MAC and MARVEL don’t explicitly represent process components in models, and are therefore not able to formally analyze and dynamically evolve process models. The feature of explicit representation of process is crucial to attain effective process management. The assessment also discovers that not all PSEE supports aspects such as product modeling, version/configuration management, transaction model, workspaces, goals for planning support. Almost all PSEE provide process modeling languages (PML) which are enactable and visualize by graphical means. In general, enormous research effort has been spent and a large number of improvements have been done to PSEE and their process support. However, it seems that there are still space for further improvement for all PSEE above in different aspects. They are e.g. integration with transaction, version and configuration management; abstraction of process models (e.g. sub-model or structural/behavior aspects); capturing and reusing past experience; sophisticated management of process models; integration with project management, etc.

In [NC94a] we have conducted in cooperation with representatives from five PSEE an assessment regarding the how they manage evolution of process models (i.e. meta-process). The five studied PSEE are SPADE, Process Weaver, ADELE, EPOS and PWI. Their brief descriptions can be found in section 2.2.2 above. We include some relevant observations below. Most PSEE concentrate on offering support at different extent for managing both static (i.e. before execution) and dynamic (i.e. during execution - on the fly) change to process models. New changes to process models are migrated to their living process instances with none or little support for impact analysis. The change propagation can be carried out either in eager (i.e. immediately visible) or lazy (i.e. visible for newly created instances) manner. This can be done by adopting either dynamic or late binding mechanism between types and instances (SPADE, EPOS); or dynamic interpreting facility of a set of rules (MERLIN); or dynamic changing the role definition/view (ADELE, PWI). Those architectural approaches seem to be appropriate to manage this kind of changes. On the other hand, none of the studied PSEE offers any support for changing to instances of process models without affecting their corresponding models. Such changes are required due to high arrival rate of unexpected events during projects, and are therefore not necessarily reflected in their corresponding process models. In addition, the studied PSEE don’t focus on recording change incidents for future projection, adjusting instances of process models to adapt to unanticipated internal and exogenous changes during project. To summarize, the studied PSEE are providing adequate support for managing planned changes to process models. They offer poor support for managing, adjusting and reacting to unanticipated changes to process instances during projects. This observation can be explained by Lonchamp’s assessment above that guiding, expressing and enforcing is most substantial concern in most PSEE. Monitoring, adapting to unexpected changes have been secondary concern.

2.3 Software Process Improvement Models

In this section some well-known methods on software process improvement are described. Software process improvement is the act of changing software development processes in such the manner that they becomes better. Such improvements are demonstrated by increasing product quality, thus customer satisfaction, productivity, and company’s profit.
2.3. Software Process Improvement Models

2.3.1 ISO 9000

It is mainly a assessment model to assure that a minimal requirement of quality system being installed in software organization. The philosophy is document what you are doing and do what you are documenting. The aim of ISO 9000 series is to achieve improvements in the quality of software products, information systems, and computer operations throughout the whole field of information technology supply including in-house development work. This is accomplished by increasing the understanding and adoption of minimum best practice as set out in the international quality standard.

The goal of ISO 9000 series is to provide guidance to companies on how to organize their quality management systems. Figure 2.1 depicts the relationship between the international standard, quality management systems, and quality plans (from [Inc94]).

![Diagram showing the relationship between International standards such as ISO 9001, Quality management system, Quality manual, and Project plans]

Figure 2.1: External standards, a quality management system and quality plan

Ince [Inc94] summarizes a quality management system as follows: This consists of the managerial structure, responsibilities, activities, capabilities and resources that ensure that software products produced by projects will have the desired quality factors that both the customer and the developer decide will be build into them.

The concrete details of a quality management system will be held in a quality manual. Such a quality manual contains standards for the quality and developmental activities that may be applied to a project together with detailed descriptions of quality control. A quality control is some activities which ensure that a particular quality factor is present in a software system and its associated documentation. In a given project, the project manager will decide on the quality controls that are needed and document this in the project quality plan. The quality plan will also describe how the quality controls are to be carried out, who will apply them, when they will be applied and other information such as whether
special-purpose tools are required.

Quality management system of ISO 9000 ensures that quality assurance activities are planned, performed, evaluated and evolved. The main responsibility of ISO 9000 will be to perform **internal quality audits**. An internal quality control is a visit to an ongoing project for the purpose of checking compliance to the quality plan. The auditing is needed because project participants tend to ignore or give low priority to quality controls.

One important criticism is that ISO 9000 ignores many of the lessons about process improvement that have been learned in other fields, specifically: learning curves. ISO 9000 ignores questions such as: How much improvement can you expect?; How fast can you expect it?; What requirements are there for improvement?; and Are there limitations on improvement?. The problem with the ISO 9000 series is that it is too general and too open for individual interpretation. This is particular a problem of confidence in the certification process, i.e. different auditors interpret the standard in different ways and certification loses its meaning.

To complement and build on ISO9000 as an International Quality Standard, the Department of Industry and Trade in the U.K. introduced **TickIT**, a quality management certification scheme specifically for products and services in information technology. The TickIT scheme was developed to certify software quality professionals to the requirements of ISO 9001 through the application of ISO 9000-3 guidance. It establishes an infrastructure for the accreditation of auditors and the recognition of the competence of auditing companies. It sets out guidelines and procedures for carrying out software audits and was established to guarantee the quality of the audit and to ensure that only software specialists carry out software quality management system audits. TickIT thus achieves a more consistent interpretation of ISO 9000 in the context of software development. As more and more countries adopt TickIT, companies wishing to export their information technology products and services will have to become aware of - and compliant with - TickIT standards.

### 2.3.2 Capability Maturity Model (CMM)

The initial objective of CMM is to assess and evaluate software capabilities of organizations for government contracts. In November 1986, the SEI, with assistance from Mitre Corp., began developing a process-maturity framework that would help developers improve their process. In September 1987, the SEI released a brief description of the process maturity framework which later expanded in Watts Humphrey's book [Hum89]. After four years of extensive experiences with preliminary version of maturity questionnaire, the SEI evolved the maturity framework into the Capability Maturity Model (CMM). The initial release of the CMM, version 1.0, was reviewed and used by software community during 1991 and 1992. Based on constructive criticisms and feedback at the workshop in April 1992, the new version of CMM, version 1.1 [PCCW93a], [PCCW93b] is released in 1993. The original objective of developing CMM was the governmental industry's need for being able to evaluate software contractor. The most recent version of CMM is described in [PWCC95].

CMM provides a framework for assessing current management and technical practices in software organization as well as improving its maturity of software engineering practices and capability to producing quality software. A maturity level is an indication or evidence for ability to develop software with a given degree of quality. Process capability describes the range of expected results that can be achieved by following a software process. It helps to predict the most likely outcome of the next software project organization undertakes.
2.3. Software Process Improvement Models

As a result of the assessment a set of recommendations aiming at improving capability are developed. Based on corporate overall goal, available resources and skill those recommendations will be prioritized and scheduled in a action plan. Companies undergo a voluntary Software Capability Evaluation to assess their maturity level, a grade from 1 to 5. Its main difference to ISO 9000 certification is that companies must climb the maturity ladder as opposed to on/off certification.

<table>
<thead>
<tr>
<th>Level</th>
<th>Process capability</th>
<th>Key Process Area</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>5. Optimizing</td>
<td>continuously improving process</td>
<td>- Defect prevention</td>
<td>Productivity &amp; Quality</td>
</tr>
<tr>
<td>4. Managed</td>
<td>quantifiable, predictable and measurable process</td>
<td>- Quantitative process management</td>
<td></td>
</tr>
<tr>
<td>3. Defined</td>
<td>standard, consistent, institutionalized process</td>
<td>- Organization process focus</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Organization process definition</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Integrated software management</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Software product engineering</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Training</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Peer–review</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Intergroup coordination</td>
<td></td>
</tr>
<tr>
<td>2. Repeatable</td>
<td>disciplined process, success dependent on individuals</td>
<td>- Requirement management</td>
<td>Risk</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Project planning</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Project tracking</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Software quality assurance</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Sub–contracting</td>
<td></td>
</tr>
<tr>
<td>1. Initial</td>
<td>adhoc, initial and chaotic process</td>
<td>- Configuration management</td>
<td></td>
</tr>
</tbody>
</table>

Figure 2.2: Five levels in Capability Maturity Model

A tabular presentation of CMM levels are depicted in figure 2.2. The presence of a set of key process areas must be demonstrated in order to be acknowledged the transition from one level to a higher one.

1. **Initial** The software process is characterized as ad-hoc, occasionally chaotic. Few processes are defined, and success depends on individual effort and heroics.

2. **Repeatable** Basic project management processes are established to track cost, schedule, and functionality. The necessary process discipline is to repeat earlier successes on projects with similar applications.

3. **Defined** The software process for both management and engineering activities is documented, standardized, and integrated into a standard software process for the organization. All projects use an approved, tailored version of the organization’s standard software process for developing and maintaining software. process capability: standard, consistent

4. **Managed** Detailed measures of the software process and product quality are collected. Both the software process and products are quantitatively understood and controlled.

5. **Optimizing** Continuous process improvement is enabled by quantitative feedback from the process and from piloting innovative ideas and technologies
CMM is essentially designated for large-scaled software organizations working with governmental contracts. An appropriate interpretation of CMM model must be in place satisfying the specific environmental context of organization whose process is assessed.

2.3.3 Quality Improvement Paradigm (QIP)

The Quality Improvement Paradigm [BR91, BCR94a] is developed at the Software Engineering Laboratory (SEL). The SEL was established in 1976 as a cooperative effort among the University of Maryland, the National Aeronautics and Space Administration/GSFC, and Computer Sciences Corporation. The main focus of QIP is to develop a scientific method for software process improvement. In this way, it can be regarded as a "meta" approach because it does not state specific technologies or areas of focus (e.g. CMM’s key process areas: configuration management, project management), but rather a method for how to find focus and how to evaluate technologies. We may say that the CMM is a top-down, while QIP is a bottom-up approach. Basili in [BCM+92] describes three concepts that support his vision of a scientific method:

- A concept of evolution The interaction and information flow between an organization and its embedded projects.
- A concept of measurement and control A structured approach to measurements including organizational goals and their empirical assessment.
- A concept of organization The introduction of a separate organizational entity to facilitate organizational learning.

The concept of evolution is supported by a loop between individual projects and the responsible organization. Such a loop consists of 6 main steps:

1. Characterization Understand the environment for better understanding based upon available models, data, intuition, etc., so that similarities to other projects and organizations can be recognized.
2. Goal setting Set quantifiable goals for successful project and organization performance and improvement.
3. Planning Choose appropriate processes for improvement, and supporting methods and tools to achieve the goals in the given environment.
4. Execution Perform the processes while constructing the products and provide real-time project feedback based on the goal achievement data for corrective actions (control cycle).
5. Analysis A post-mortem analysis of collected data and experience is carried out. The analysis will evaluate the current practices, determine problems, record findings, and make recommendations for future improvement.
6. Packaging The experience gained in the form of updated and refined models and other forms of structured knowledge are packaged into a repository for future use (capitalization cycle).

The concept of measurement and control is supported by the Goal-Question-Metric (GQM) approach [BCR94b] to structure measurement. GQM defines a measurement model on three levels:
1. **Conceptual level - Goal** A goal is defined for an object, for a variety of reasons, with respect to various models of quality, from various points of view, and relative to a particular environment.

2. **Operational level - Question** A set of generic questions is used to define models of the object under study and then focus on that object to characterize the assessments or achievements of a specified goal.

3. **Quantitative level - Metric** A set of metrics, based on the models, is associated to every question in order to answer it in a quantitative way.

The underlying philosophy is that one should not measure anything without knowing exactly how to use the results. GQM can be seen as a method to define measurements in a way that document their intentions, as well as how they should be interpreted. However, GQM does not give any indication of the phenomena that can be measured, how it is done, or how to assess the validity of the measurement.

The concept of organization is supported by the Experience Factory (EF) [Bas93, BCR94c]. The EF, being an independent organizational unit, is stated to institutionalize the collective learning of the organization that is at the root of continual improvement and competitive advantage. The EF seems to be a repository with reusable information, organized as a separate organizational entity. The data that can be compared between projects are those that constitute the baseline. The SEL baseline includes effort distributed on phases and activities. The EF comprises of activities in analyzing and packaging experiences. **Experience base** is a repository storing software engineering models (of process, product, resource, quality, defect, …) and other relevant knowledge.

The QIP’s 6 steps together with the independent organizational unit EF are depicted in figure 2.3.

![Figure 2.3: QIP and Experience Factory](image)

**SEL Process Improvement Paradigm**

Another variant of QIP is **SEL Process Improvement Paradigm**, described in [MPP+94]. This is considered as an example of operational experience base. The main focus is to understand and improve
current practices by experimenting with new technologies; getting more empirical data on their impacts; refining/tailoring them for specific needs; and institutionalizing them as company-wide standard. The paradigm identifies following architectural agents: Technology Advocates (TA); Experimenter (E); Model Packager (MP); Designer/Developer (DD). TA has access to existing technologies on the field of interest and receives a description of problem identified by DD. TA will then propose a technology candidate for further experimentation to the E who carries out experiment on a pilot project, collects measures and then sends synthesized information to MP. The refined technology will be packaged by means of reports, standard, tool with which DD uses for their current process. Problem findings will be reported back to TA.

The SEL Process Improvement Paradigm includes following steps:

**Understanding:** characterizes the current practices of software process within an organization. The local baseline is then established and clear and quantitative improvement goals are defined with respect to process, product quality (corresponding to steps 1 and 2 in QIP).

**Assessing/Refining:** evaluates existing technologies and methods and then determines the trial candidate for further experimentations within a small controlled environment. The hypothesis is that application of the trial technology to current development/maintenance environment will yield overall beneficial advantages. The results are measured according to the improvement goals set by previous step. The collected measures are basis for analyzing and assessing the impact of trial technology on the improvement of current process comparing with the initial state. The applied technology can be adjusted to accommodate organization’s needs and characteristics (corresponding to steps 3 and 4 in QIP).

**Packaging/Infusing:** the positive results achieved and lesson learned will be packaged by means of experience reports, handbook/procedure, corporate-wide standard/policy for doing business, tools, process models for future similar projects. The new technology is trained for personnel and infused into production environment (corresponding to steps 5 and 6 in QIP).

Software Management Environment (SME) is an automated tool offering support for accessing/extracting relevant information from experience base for comparing, predicting and assessing ongoing projects with previous projects having similar characteristics. Typical metrics which are collected relate to i) *process*, including effort distribution, language usage, software growth profile, change profile, process model, tool, etc; ii) *cost*, including productivity (code rate), maintenance, cost per change (e.g. specification changes), level of rework; iii) *product*, including size, code reuse rate; iv) *reliability*, including class of errors (error source), origin of errors (error distribution), error detection (error rate).

### 2.3.4 Bootstrap - An Assessment Method

BOOTSTRAP [HMK+94] is a European assessment methodology developed by an ESPRIT project [KBC+93] for software process assessment and improvement between October 1990 and February 1993. BOOTSTRAP has been build on the basis of the SEI’s CMM (in section 2.3.2), the ISO 9000 series (in section 2.3.1), and the European Space Agency’s generic software process model.

In 1993 the project partners formed the BOOTSTRAP Institute to continue the development and transfer of the methodology into the software industry. BOOTSTRAP is also on of the main background
2.3. Software Process Improvement Models

methodologies of the emerging ISO standard for software process assessment, better known as SPICE (Software Process Improvement and Capability Determination - described in section 2.3.5 below). As a result of the SPICE work, the new version of the BOOTSTRAP methodology is SPICE conforming.

Main features of the method are:

- It provides detailed capability profiles and maturity levels.
- It gives profiles and levels of organizations and projects separately.
- It allows benchmarking, i.e. the comparison with industry, country, European and other assessment data stored in the BOOTSTRAP database.
- It allows one to determine the degree of fulfillment of the the ISO 9000 standard elements.
- It also allows one to calculate the maturity level according to the SEI’s CMM method.
- It includes immediate feedback from the organization assessed.

With BOOTSTRAP, an organization and its process quality is assessed with respect to organization (O), methodology (M) and technology (T), divided into the process attributes shown in figure 2.4. The concept underlying BOOTSTRAP is that an organization has to fulfill basic organizational requirements first, and then to care about methodology before making investment in technology as third priority.

The BOOTSTRAP framework is an attribute-based method for assessing process maturity and constituting a foundation for manually making improvement action plan after using the BOOTSTRAP profiling technique. BOOTSTRAP’s primary objective is developing a method for software-process assessment, quantitative measurement, and improvement. It enhanced and refined the SEI’s CMM assessment method and questionnaire to adapt it to the needs of the European software industry.

The BOOTSTRAP method has three main components:

1. Quality attribute: An overall corporate goal is decomposed to several quality attributes which in turn are assigned by hierarchy of quality factors. Three main dimensions of process are focused: organization, methodology and technology. Those three aspects are represented as three main quality attributes clusters in Bootstrap’s hierarchy process-quality attribute.

2. Refined algorithm: of SEI on determining maturity level. BOOTSTRAP method calculates a maturity level for each attribute of the process quality model by assigning a software metric to each attribute and evaluating it to get a set of measured values that represent process quality.

3. BOOTSTRAP questionnaires: Each quality attribute is associated to a checklist which together constitutes Bootstrap questionnaires. The degree of satisfaction of the answers decide process maturity level from 2 to 5.

The BOOTSTRAP Methodology is fully aligned with ISO 9000 and is consistent with the CMM. However, the BOOTSTRAP Methodology provides important profiles detailing the maturity of each major aspect of software development both at an organization and individual project level.
2.3.5 Software Process Improvement and Capability Determination (SPICE)

In order to provide the Software Community with a universal model for process assessment, ISO started to analyze the requirements for a standard in June 1991. The SPICE international project [Dor93] has been initiated to develop the products (process model, procedures and documents) concerning the future ISO standard.

The SPICE project provides a framework for the assessment of software processes. This framework can be used by organizations involved in planning, managing, monitoring, controlling, and improving the acquisition, supply, development, operation, evolution and support of software. SPICE embodies a sophisticated model of software process management drawn from the world-wide experience of large and small companies. SPICE provides:

- a framework for determining key process weaknesses;
- the understanding necessary to improve the software process (i.e. to define areas and priorities for software process improvement), and to measure such improvement;
- the means by which purchasers of software can determine the capability of competing suppliers to deliver products and services which are fit for purpose and on time.
- an understanding of the risks for a business considering development of a new software product or service.

The framework for process assessment:
2.3. Software Process Improvement Models

- facilitates self-assessment;
- takes account of the context of the process being assessed. The sophistication and complexity required of a process depends on such a context;
- produces a process rating profile rather than a pass/fail result to a process instance. A process instance is a singular instantiation of a process that is uniquely identifiable and about which information can be gathered in a manner that provides repeatable ratings. Each process instance is characterized by a set of five process capability level ratings, each of which is aggregation of the practice adequacy ratings that belong to that level;
- addresses the adequacy of practices relative to the process purpose. Practice adequacy is a rating of the extent to which a practice meets its purpose. From the ratings of process instances, a number of derived or average ratings can be determined to provide capability of a process within an organizational unit as a whole;
- is appropriate across all application domains and sizes of organization.

![Diagram of the SPICE assessment framework]

Figure 2.5: The SPICE assessment framework

SPICE’s assessment model comprises of five process categories, such as customer-supplier relation; engineering process; project management process; support; and organization. In each process category, the capability is determined by five levels.

SPICE process assessment provides users with the ability to evaluate process capability on a continuous scale, rather than using the pass/fail characteristic of quality audits based on ISO 9000. In addition, the SPICE framework provides the opportunity to adjust the scope of assessment to cover specific processes of interest, rather than all of the processes used by an organizational unit.

2.3.6 Discussion

All those software process improvement models above share the same objective or philosophy - to make the software process better. The gained improvement is demonstrated by increasing customer
<table>
<thead>
<tr>
<th></th>
<th>ISO 9000</th>
<th>CMM</th>
<th>QIP</th>
<th>Bootstrap</th>
<th>SPICE</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Purpose</strong></td>
<td>achieve certification by conforming to an international standard</td>
<td>stepwise and rigorous roadmap for improving process maturity and capability</td>
<td>process improvement by experimenting with new technology, and learning from past experience</td>
<td>less rigorous than CMM based on hierarchy of key quality attributes</td>
<td>customize assessment based on context and objective to determine capability</td>
</tr>
<tr>
<td><strong>Focus</strong></td>
<td>conformance, certification</td>
<td>assessment, maturity and capability determination</td>
<td>experimentation, experience reuse and learning</td>
<td>assessment, benchmarking, improvement plan</td>
<td>local assessment, narrowed focus/ context, improvement plan</td>
</tr>
<tr>
<td><strong>Strategy</strong></td>
<td>document what to do and do what is documented</td>
<td>demonstrate the presence of KPA defined for each level</td>
<td>choose appropriate technology for experiment to solve the identified problem</td>
<td>assess process status; compare with industry; prioritize and elaborate improvement plan</td>
<td>assess a rather narrowed process scope, and make improvement plan</td>
</tr>
<tr>
<td><strong>Concept</strong></td>
<td>- Quality MS</td>
<td>- Maturity level</td>
<td>- no rigorous framework to assess and then identify weakness</td>
<td>- Organization</td>
<td>- Customer/supplier</td>
</tr>
<tr>
<td></td>
<td>- Quality manual</td>
<td>- Key process area</td>
<td></td>
<td>- Methodology</td>
<td>- Engineering process</td>
</tr>
<tr>
<td></td>
<td>* standard</td>
<td>- Key practices</td>
<td></td>
<td>* engineering support</td>
<td>- Project management</td>
</tr>
<tr>
<td></td>
<td>* procedure</td>
<td></td>
<td></td>
<td>* product engineering</td>
<td>- Support</td>
</tr>
<tr>
<td></td>
<td>* quality control</td>
<td></td>
<td></td>
<td>* process</td>
<td>- Organization</td>
</tr>
<tr>
<td></td>
<td>* Quality plan</td>
<td></td>
<td></td>
<td>* Technology</td>
<td></td>
</tr>
<tr>
<td><strong>Method</strong></td>
<td>determine pass/fail by auditing the Quality Management System</td>
<td>determine achievement of a maturity level by present/absent of KPA</td>
<td>Goal/Question/Metric; Experience Base Measurement</td>
<td>rating process in a continuous scale</td>
<td>rating capability in a 5–level scale</td>
</tr>
</tbody>
</table>

Table 2.1: Comparison of five software process improvement models

satisfaction and company profits. The ISO and CMM focus on conforming software process to an international and rigorous standard, i.e. a normative model based on currently best practices in industry. The QIP, Bootstrap and SPICE emphasize to locally assess the current practices, identify weaknesses and then elaborate or customize a local improvement plan. Although all models are required to be customized to particular organization, we can regard the former case as top-down approach, and the latter case as bottom-up approach. Such classification reflects how the improvement initiatives are originated, i.e. by strictly following an international standard or by locally identifying existing deficiencies and then prioritizing corrective actions. SEI’s CMM is a descriptive approach illustrating the normative behavior of software organization without a clear method (i.e. which technology is required to demonstrate the presence of a particular KPA) on how to obtain the desired maturity level. It requires a correct interpretation, appropriate tailoring and consistent implementation of the standard CMM to specific needs of local organization. There exist a lot of software process improvement initiatives other than those described above. Still, not many software organizations are willing to implement improvement program. The scepticism is caused by the fear of a huge effort spent and little gain and benefit in return. It is therefore important to direct more focus on conducting several empirical studies in industry to provide quantitative evidences on return of investment before the improvement models can gain common acceptance and widely used. In addition, the improvement initiatives may be scaled down in size and and complexity to better suit small and medium enterprises.

The comparison of five software process improvement models is illustrated in table 2.1. We have defined five criteria to compare them such as: general purpose; main focus; strategy; assessment concept and method to be adopted.
2.4 Software Evolution

2.4.1 What is Evolution?

The Merriam-Webster's Collegiate Dictionary has following definitions for evolution: 1) a process of change in a certain direction; 2) a process of continuous change from a lower, simpler, or worse to a higher, more complex, or better state; 3) a process in which the whole universe is a progression of interrelated phenomena.

According to Lehman in [Leh94], the term evolution is used to describe a phenomenon observed in a variety of entity classes, each in its own context. Natural species, societies, cities, artifacts, concepts, theories, ideas are all said to evolve. The term describes a process of progressive change in the attribute of the entity(ies). Such change may include e.g. improvement in some sense, adaptation to changing environment, loss of not-required or undesired properties or the emergence of new ones.

2.4.2 Evolution in Software Technology

In 1969, Lehman examined and modeled the continuing change process being applied to IBM's OS/360 operating system. This investigation has derived models of that system's evolution from measures of software release characteristics and used them to support planning and management of sequences of such releases. In collaboration with Belady, he postulated and investigated a phenomenon initially termed Program Growth Dynamics and later renamed Program Evolution Dynamics. It was probably this study that first applied the term evolution to a phenomenon in the field of software technology. This study has yielded valuable insight into the evolutionary nature and properties of software and its associated process.

We will present and discuss some models of software evolution to get a better insight into why software and its process change over time. In the two following sections, we will present two models of software evolution representing a product and a process oriented view of evolution. The product oriented model is the SPE-classification of Belady and Lehman [LB85]. The focus of their model is that different program types will have different evolution patterns according to the type of problem the program is designed to solve. The data collected from several studies have provided strong evidence that software evolution is a feedback phenomenon. Lehman thus stated that program evolution process is a complex multi-agent, multi-level, multi-loop feedback system. The process oriented model present an approach to control the evolution of software process by categorizing process changes by properties, and controlling the implication of these changes based on a model of the items to be changed and their interdependencies.

2.4.3 Program Evolution Dynamics

2.4.3.1 SPE classification

As mentioned above, software varies widely. To distinguish between such changes, one may start by classifying the entities according to their evolutionary characteristics [Leh94]. By studying the common factors and distinctions between classes one learns more about each class and obtains clues to the
relationship between their evolutionary and other properties and how individual evolutionary patterns may be directed, controlled, or predicted.

Lehman and Belady in Chapter 20 in [LB85] had therefore observed that computer programs can be classified into three different groups with respect to software evolution. This classification is known as the SPE program classification.

**S-type program**

Programs in this category are formally defined and derivable from a specification. This category contains all the programs that are solutions to a specific problem. Examples are to compute the square root of an integer, sort a sequence of numbers, etc. Each problem is defined by its specification, and this specification with respect to the identification of the problem is correct. The resulting program may have an interest in the universe of discourse. Figure 2.6 depicts this type of programs.

![Figure 2.6: S-type program [LB85]](image)

**P-type program**

Programs in this category are solutions to a real-world problem. Compared to S-type programs, problems in this category are hard. Complete solutions are not practical due to the size of the problem, or its complexity. Examples of P-type programs are a weather forecast system, and a chess playing program. This chess program may be programmed to contain data of all possible games of chess in a tree data structure; this would ensure that the chess program were unbeatable by humans. Such a solution is however not practical; instead a set of rules for playing chess can be specified and coded into the program, with abilities to learn from past mistakes. Since the problem is too complex to be solved mathematically correct, an approximation to the correct solution can be specified. Figure 2.7 depicts this type of programs.

The major difference between S- and P-type program is expressed by the comparison cloud in figure 2.7. In S-type programs, the correctness of the program is determined only by its specification. For P-type programs, the applicability of the solution is validated in its real-world context. Lehman and Belady in [LB85] states that:
“Differences between data derived from observation and from computation may cause changes in the world view, the problem perception, its formulation, the model, the program specification and/or the program implementation. Whatever the source of the difference, ultimately it causes the program, its documentation or both to be changed. And the effect or impact of such change can not be eliminated by declaring the problem to be a new problem, for the real problem has always been as now perceived. It is the perception of users, analysts and/or programmers that has changed.”

“But the world too changes and such changes result in additional pressure for change. Thus P-type programs are very likely to undergo never-ending change or to become steadily less and less effective and cost effective.”

**E-type program**

Programs in this category provide automated assistance to a human or social activity and thus are embedded in the operational environment, implementing an application in that environment. E-type systems have no direct boundaries. The programs that implement them cannot have permanent and demonstrably satisfactory specifications since the variety of features that can be built into such systems are unlimited. The system is a “bounded and discrete model of an unbounded and effectively continuous application domain” [Leh94]. When the developers decide what features to include into the specification, they must take into account the perceived needs of the users, available technology, and non-functional properties as performance and cost. Unlike S- and P-type programs, E-type program changes the environment, since the system is introduced as a component in its environment. Future systems in the same environment must treat this system as part of their real-world.

Examples of E-type programs are operating systems, air-traffic control systems, office automation systems, and integrated programming support environments (IPSEs). Being part of the environments, E-type programs will inevitably be part of the changes in the environments, as well as users will propose changes as the limits of the current implementation are realized through extensive use. Figure 2.8 de-
picts the E-type program. Lehman and Belady in [LB85] states that:

![Diagram of the E-type program]

Figure 2.8: E-type program [LB85]

Lehman and Belady in [LB85] states that:

"As they become familiar with a system whose design and attributes depend, at least in part, on user attitudes and practice before system installation, users will modify their behavior to minimize effort or maximize effectiveness. Inevitably this leads to pressure for system change. In addition system exogenous pressures will also cause changes in the application environment within which the system operates and the program executes. New hardware will be introduced, traffic patterns and demand change, technology advance and society itself evolve. Moreover, the nature and rate of this evolution will be markedly influenced by program characteristics with a new release at intervals ranging from one month to two years, say."

2.4.3.2 Laws of Program Evolution

Lehman and Belady ([LB85], p.381, p.412) also postulates five laws of program evolution, based on observations on several systems evolving over a series of releases (cf. IBM's OS/360).

1. Law of Continuing Change: A program that is used and that is an implementation of its specification reflects some other reality, undergoes continual change or becomes progressively less useful. The change or decay process continues until it is judged more cost effective to replace the system with a recreated version.
2.4. Software Evolution

2. Increased Complexity: As an evolving program is continually changed its complexity, reflecting deteriorating structure, increases unless work is done to maintain or reduce it.

3. The Fundamental Law of Program Evolution: Program evolution is subject to a dynamics which makes the programming process, and hence measures of global project and system attributes, self-regulating with statistically determinable trends and invariance.

4. Conservation of Organizational Stability: During the active life of a program the global activity rate in the associated programming project is statistically invariant.

5. Conservation of Familiarity: During the active life of a program the release content (changes, additions, deletions) of the successive releases of the evolving program is statistically invariant.

2.4.3.3 Feedback driven Evolution Process

Process evolution in Lehman's terminology includes changes in software system, software process and support environment in both development and maintenance phase. He characterizes operational domain in which software system is intended to support or solve a specific problem as a closed loop structure in control theory. That is, any changes are triggered and driven by a feedback or learning process in which people and organization extend their knowledge and understanding. All changes must be understood and mastered if software evolution is to be directed, planned and controlled. Lehman has in [Leh94] identified 5 classes of evolving entities:

1. Release or a sequence of versions of a program or software system. Changes to such releases are needed to correct errors and faults, provide support for new hardware, improve performance and cost-effectiveness, increase functionality and so on. Relevant information is fed back to supplier and demands correction. In this feedback, the people involved have a direct impact on the information, that is on feedback characteristics. The source of feedback is not only confined to developer and user community, but also many others contribute to the change process. It is the long external user and business based feedback loops that are primarily responsible for the characteristics of release dynamics.

2. Software system or a program. A program is developed based on a set of application concepts or problem statements. As initial requirements changes and technology evolves, the software system must also undergo a series of transformations triggered by several feedback driven iterations. For a S-type program whose specification is originally complete, the evolution is driven by feedback from unsatisfactory output to indicate changes to the specification. On the other hand, for a E-type program whose specification is not initially fixed, the evolution feedback can not be determined before program execution. The evolution of software system is influenced by the internal feedback.

E-type application It relates to the unbounded nature of E-type application, and the unbounded nature of the domain over which the application is to be valid or supported. Once the system is in operation, the need to extend its use to regions of the domain previously excluded will inevitably arise. The system must thus be evolved to satisfy this newly recognized or emerging need. This is achieved by feeding back to the developmental organization relevant information. The application, its domain and the system comprise a multi-loop, stochastic, non-linear feedback system.
whose evolution is the consequence of the individual but interacting evolutionary behavior of its constituents.

**Process of software evolution** considering both development and maintenance process. Unanticipated circumstances and unexpected conditions often lead to process adjustment, adaptation and changes on the fly. The consequence may be a change to the planned upcoming process activity or a need to backtrack or iterate. In any event there is a complex mixture of feedback and feed forward. The process evolution is dominated by internal feedback primarily to analysis of the product and the process that produces it.

**Process models** Models, preferably formal, are essential as vehicles for communication and reasoning. The real world is represented by models which capture the domain of interest. Such models are imprecise, incomplete and uncertain due to their embedded assumptions. Thus, if such a model is to serve an useful purpose it must continually reflect the process as it evolves. The evolution of a process and its model must be linked. The information that drives evolution, stems from observation and previous experience. Model evolution is also feedback. The flow will be from within the organization, from other software developers and from process experts, software engineers [Leh91].

### 2.4.4 PRISM - A Model of Change

In [Mad92] Madhavji has identified various *items* strongly related to software development environment which are subjected to continuous change. The basic items of change are people, policies, laws, resources, processes, and results. For various predictable and unpredictable reasons, such items may need to be changed on an on-going basis. Therefore, the PRISM which is a model of change, was designed and included following key or unique features:

- separation of changes to the described items from the changes to the environmental facilities encapsulating these items.
- An infrastructural facility, called the *Dependency Structure*, describes various items, and their interdependencies. Such a structure is used to identify the items affected by a given change.
- An infrastructural facility, called the *Change Structure* classifies, records and analyzes change-related data, and makes qualitative judgment of the consequences of a change.
- identification of the many distinct properties of a change.
- a built-in mechanism for providing feedback.

The PRISM model of changes incorporates a two-tier approach to designing, implementing, and controlling changes to the items in an environment. The first aspect deals with identifying the set of items affected due to a given change; and the second deals with classifying and recording analytical data related to the change. These aspects also include feedback mechanisms to instigate changes to items and provide projection for future changes.

Figure 2.9 shows two levels of operation in PRISM model. At the normal level, there exists the use of the two environment infrastructures, called Dependency Structure and Change Structure. At the meta
level, there is a build-in mechanism for supporting (meta) changes to two environment infrastructure.

![Diagram showing the PRISM model of change (Mad92)](image)

**Dependency Structure**

Dependency Structure explicitly represents basic items of changes (people, policies, laws, process, resource and result) in different types and their interdependencies. These types can occur in six levels of abstraction, such as Community, Nation, Organization, Project, Group and Individual (listed in downward order). Together, the types of items and the levels of abstraction can represent a comprehensive set of items concerning a software project. This has been kept implicit in the model so as to maintain simplicity. Changes can be propagated downwards and feedback information can be propagated upwards. Changes and feedback can also be propagated within a level of abstraction. Such propagations are carried out in a structured way with defined change/feedback interface specifications. At a given level of abstraction, there are a set of tightly related items of change. Each such set is called a *sheet*. In a software environment there are usually many interacting sheets, forming a system of sheets or representing software processes. In such a sheet system, various change items encapsulated within a sheet can be exported to affect, or impose upon sibling and lower level sheets. Similarly, feedback concerning the use of an imported item can be propagated from a lower level sheet to other sibling or higher ones.
Change Structure

Change Structure facilitates classification, recording, and analysis of change-related data. The Change Structure provides three important features:

1. Integration with the Dependency Structure permits switching back and forth between an item of change in the Dependency Structure and its change-related data in the information base (see figure 2.10). A change item declared in the Dependency Structure is associated with a set of components in the Change Structure, each of which is called a CS-sheet. A CS-sheet describes a particular change in the history of change to a given item. Every time a change is made to an item in a DS-sheet, all the details of the change and related data are recorded in a new CS-sheet associated with the item. During the process of making a change, it may be necessary to access previously recorded history of changes made to the item.

2. Framework of change is a classification scheme which simplifies the structuring of the decision made for a given change to an item. The framework for a change is quintuple: i) type of the item of change including one of people, policies, law, processed, resources, and results; ii) location of the item in the environment, i.e. precise DS-sheet in the Dependency Structure where the item is declared; iii) DS-feedback concerning the item that is gathered in the Dependency Structure and that which instigated the current process of change; iv) properties of a change, including a decision of whether to make a proposed change, and person responsible for the change; v) substructure of a change including categories: predicted and actual. The change properties and associated substructures are kept implicit in the model so as to maintain simplicity. Table 2.11 gives a list of many properties of change. This list is not meant to be exhaustive, and for the construction of an operating Change Structure there would be a need to extend this list.

3. Information base provides the necessary data to support the process of making change-related decisions for subsequent changes. The information recorded must cover: i) all the factors that affect a given change, and ii) the consequences of this change.

Infrastructural Changes

The meta-feedback in figure 2.9 which can originate in any DS-sheet, or CS-sheet, at any level of abstraction, is continuous during the life cycle of a project. An important question that needs to be addressed is how to incorporate any infrastructural changes. Two of the key issues involved in this problem are:

1. the necessity of transferring change information from the old DS-sheets and CS-sheets into the corresponding new sheets. That is, the compatibility between the old and new versions of the infrastructures is maintained.

2. the methodology for carrying out a transfer if necessary. This issue deals with two approaches: i) eager instantiation, implying instantiation new sheets as soon as the new infrastructure is ready for release; and ii) lazy instantiation, meaning instantiation a new sheet upon demand.
2.4. Software Evolution

Figure 2.10: Integration of CS-sheets with a DS-sheet ([Mad92])

<table>
<thead>
<tr>
<th>Criteria for deciding whether to make the change</th>
<th>Structure of change process used: meta life cycle</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source of change</td>
<td>- requirement, specification</td>
</tr>
<tr>
<td>- local feedback in DS (eager change)</td>
<td>- design and coding</td>
</tr>
<tr>
<td>- non-local feedback in DS (demand change)</td>
<td>- testing and simulation</td>
</tr>
<tr>
<td>Decision of whether to make the change</td>
<td>- manufacturing</td>
</tr>
<tr>
<td>Advantages of carrying out the decision taken</td>
<td>- maintenance</td>
</tr>
<tr>
<td>Disadvantages of carrying out the decision taken</td>
<td>- Method for carrying out the change</td>
</tr>
<tr>
<td>Reason for the change or non-change</td>
<td>- batch style, incremental style</td>
</tr>
<tr>
<td></td>
<td>- Resources needed to make the change</td>
</tr>
<tr>
<td>What aspect of the item the decision is about</td>
<td>- Person responsible for the decision</td>
</tr>
<tr>
<td>Type of change</td>
<td>- Reviewer of change</td>
</tr>
<tr>
<td>- static, dynamic</td>
<td>- quality control person, process program</td>
</tr>
<tr>
<td>- corrective, adaptive, perfection</td>
<td></td>
</tr>
<tr>
<td>Size of change (to this item)</td>
<td>- Reliability of the change process</td>
</tr>
<tr>
<td>Impact of a change on the entire system</td>
<td>- Repeatability of the change process</td>
</tr>
<tr>
<td>- qualitative, quantitative along with</td>
<td>- Extensibility of the change process</td>
</tr>
<tr>
<td>- the degree of each</td>
<td>- including contractibility</td>
</tr>
<tr>
<td>Cost</td>
<td>- Re-usability of the change process</td>
</tr>
<tr>
<td>- of obeying decision</td>
<td>- portability, generality supports</td>
</tr>
<tr>
<td>- of not obeying decision</td>
<td>- reusability</td>
</tr>
<tr>
<td></td>
<td>- Efficiency of the change process</td>
</tr>
<tr>
<td>Quantitative coupling with other changes</td>
<td></td>
</tr>
<tr>
<td>Qualitative coupling with other changes</td>
<td></td>
</tr>
<tr>
<td>Relation with a previous change</td>
<td></td>
</tr>
<tr>
<td>- how new is this change</td>
<td></td>
</tr>
<tr>
<td>Explicit constraints acting on the change</td>
<td></td>
</tr>
<tr>
<td>- policy, ley</td>
<td></td>
</tr>
<tr>
<td>- sequencing of changes of activities</td>
<td></td>
</tr>
<tr>
<td>- maximum, minimum changes permitted</td>
<td></td>
</tr>
<tr>
<td>- cost, method, etc.</td>
<td></td>
</tr>
<tr>
<td>Compromises made in the change</td>
<td></td>
</tr>
<tr>
<td>- includes agreements with other</td>
<td></td>
</tr>
<tr>
<td>Conflict caused by the change</td>
<td></td>
</tr>
<tr>
<td>- other people</td>
<td></td>
</tr>
<tr>
<td>- functioning of other processes,...</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Phase of the project in which the decision is made</td>
<td></td>
</tr>
<tr>
<td>Schedule of the change</td>
<td></td>
</tr>
<tr>
<td>- of the review process for this change</td>
<td></td>
</tr>
<tr>
<td>Anticipation of the change</td>
<td></td>
</tr>
<tr>
<td>- anticipated, unanticipated</td>
<td></td>
</tr>
<tr>
<td>Ease of change</td>
<td></td>
</tr>
<tr>
<td>- trivial, ... not possible</td>
<td></td>
</tr>
<tr>
<td>Absorption of change</td>
<td></td>
</tr>
<tr>
<td>- how much change can be sustained by item</td>
<td></td>
</tr>
<tr>
<td>Reversibility of the change</td>
<td></td>
</tr>
</tbody>
</table>

Figure 2.11: Example properties of a change ([Mad92])
2.4.5 Discussion

The initial work of Lehman has undoubtedly revealed the evolutionary nature and its consequence of both software system and its development processes. This work has certainly motivated research community to find effective methods for achieving better control over evolution. Lehman’s work has focused on classification of evolving software entities into classes. By doing so, common factors and distinctions between classes provide better understanding of the patterns which may be used to direct, control or predict in similar circumstances. This work is thus characterized as being oriented to item of change. Among other valuable lessons we learn from this work is that the ability to predict can be gained by formal classification and projection. On the other hand, we perceive another means to manage evolution by focusing on the change incident itself. It means that arbitrary change incidents are captured, categorized and associated to the project context in which changes are detected. The same philosophy of Lehman’s work is still applicable in such the sense that changes incidents can be grouped into common patterns which then relate to particular project characteristics, in order to direct, control and anticipate future projects.

Such process change-oriented perspective is satisfied by the PRISM model of Madhavji. PRISM offers a framework and associated infrastructures to represent dependencies between evolving entities, classify and record changes together with their properties. The model also defines a feedback-based approach to evolve the infrastructural facilities. Each item of change is associated with a set of previous change histories which record a large number of related informal data about change properties (as depicted in figure 2.11). The items of change occurring in different levels of abstractions, interact tightly to each other to represent software process models by means of a sheet program. In general, the PRISM model is well defined and comprehensive to deal with change management in software project. However, the number of change properties and their associated values are unlimited and freely and heuristically specified. It therefore implies that such recorded change properties are hard to be formally represented, thoroughly analyzed and reasoned about in the information base. As a consequence, it is difficult to group or recognize change incidents into typical patterns. On the other hand, the model doesn’t define any characteristics of the basic items of changes associated to a given change incident. Such relationships are crucial when predicting what kinds of change likely occur based upon a set of external characteristics of a particular item of change. The model also lacks of an operational approach to exploit change-related data to support future projection and decision making process concerning similar changes. The model infrastructural facilities (i.e. Dependency Structure and Change Structure) are only defined at a methodological level, and thus not properly integrated into a PSEE.

2.5 Software Measurement

Software development organizations begin to recognize that measurement is a prerequisite for systematic process improvement, and have started to measure their products and processes in order to understand, analyze, plan, and guide their projects [LHR95]. Software measurement is essential to conduct empirical study of software engineering process in order to achieve a quantitative foundation for making improvement. We will therefore devote this section to briefly discuss the fundamental theory behind software measurement. Some typical PSEEEs which have integrated measurement capabilities to improve and guide software processes, are presented.
2.5. Software Measurement

2.5.1 Basic Measurement Theory

In [Fen94] measurement is defined as the process by which numbers or symbols are assigned to attributes of entities (e.g. person, document, process) in the real world in such a way as to describe them accordingly to clearly defined rules. Attributes of entities can be perceived differently by different people. A model, reflecting a specific view, must be defined for the object being measured.

- **Direct measurement**: of an attribute is measurement which does not depend on measurements of other attributes.

- **Indirect measurement**: of an attribute is measurement which depends on one or more other attributes.

Measurement is used either for assessment or prediction. The prediction system consists of a measurement model; a procedure for determining model parameters; and a interpretation of results. Empirical Relation System \((C,R)\) is a pair of entities \(C\) and a set of empirical relations \(R\). Representation \(M\) is a function mapping from \((C,R)\) to a numerical relation system \((N,P)\) while the empirical relations are preserved. In [Fen94] validating a software metrics in the assessment sense is equivalent to demonstrating empirically that the representation condition is satisfied for the attribute being measured. A metric in predictive sense is validated if the prediction system must be clearly specified and a proper hypothesis proposed, before experimental design for validation can begin.

Also in [Fen94] software engineering, three entities are of primary interest: process, product and resource whose attributes are distinguished as follows:

- **Internal attribute**: can be measured purely in terms of process, product or resource itself (e.g. effort, size, cost)

- **External attribute**: can be measured with respect to how process, product or resource relates to other entities in environment. (productivity, reliability, profitability).

[BBF+90] exploits the usefulness of measures to make precise estimations. Four steps in building software measurement:

1. attributes of software products and processes should be well understood.

2. formal models or abstraction that capture the attributes to be measured.

3. important relationships and ordering that exist between the objects must be defined. It means thus that a partial ordered relation between two attributes must be elaborated to be able to compare and to make the common norm.

4. mapping from the models to number systems that preserve the order relationships (metrics)

There are two types of metrics which are classified according to the time they are collected. They are early life cycle metrics and code metrics. The former type of metrics is usually utilized as basis or indicator to predict the future behavior, while the code metrics are used to make assessment against goals or objectives. The early life cycle metrics are derived from either specification or design documents.
They are e.g. Function point [AJ]; DeMarco’s Bang metrics [DeM82] based on formal specification notations; Ripple effect [YCM] in maintenance activity; Cluster analysis [HB]; and Information flow analysis [HK]. The code metrics are e.g. Halstead’s software science [GH]; McCabe’s complexity metric [McC] based on control flow; and Oviedo’s hybrid metrics [Ovi] on both control and data flow.

### 2.5.2 Measurement-based PSEEs

Measurement requires a solid understanding of the process, product and resources to be measured. That is, such measurable entities must be explicitly modeled. Among other objectives of a PSEE is to explicitly and formally model entities mentioned above. Therefore, it is important to integrate modeling and measuring capability to effectively guild process performance. Below is a description of some Process-centered Software Engineering Environments (PSEE) applying measurement to improve and guide software development process.

#### 2.5.2.1 Ginger

In [iMKKT93] Matsumoto et al. has proposed a process improvement framework based on modeling and measuring processes that is supported by the Ginger system. Ginger is an environment that collects nontrivial product measures automatically and uses these measures as simple process guidance. Process data, such as the amount of computer-usage time and the commands that were executed, are collected by the supporting operating system. Product data, such as source-code files and changes to code, are automatically collected at regular intervals by the environment. Ginger consists of a set of monitoring and feedback tools designed to record and improve programmer productivity during coding activities. Data is collected from on-line activities unobtrusively and automatically, and this data is fed back to the programmer and manager upon request in real-time. An user can then compare his/her performance with baselines for his/her organization and receive immediate feedback based on the models supported in the environment. Some experimental studies has been conducted and provided significant evidences on the increased efficiency in carrying out software development with such measurement-based guidance. Measurement support is fixed, and Ginger does not support process definition.

#### 2.5.2.2 Amadeus

Amadeus is an integral part of the Arcadia SEE [TBC+88] whose focus is on measurement and capability to be integrated with a PSEE. In [SaPSB91] Selby and Porter has reported their effort on separating the concerns of measurement from those of process guidance. Their major focus has been solely providing measurement services for a PSEE. Amadeus offers extremely flexible ways to specify the collection and evaluation of measurement data. Such a feature is realized by providing an interface that makes it possible to integrate measurement capabilities into any PSEE. During process enactment, on-line mechanisms recognize triggers, and respond by invoking tools to collect and analyze data. It is helpful to think of Amadeus as a library of services for performing measurement and generating feedback based on the valued collected. However, Amadeus does not directly support process modeling or enactment.
2.5.2.3 ES-TAME

Ovio and Basili developed the ES-TAME system [OB92] to illustrate evidences of the concept and their experience base idea in QIP and G/Q/M and Experience Factory. Their experience base is a repository for organizational knowledge about developing and maintaining software products, which is expressed in terms of process, product and quality models. The ES-TAME system offers a methodology, knowledge representation, and reasoning framework for manipulating the information in the experience base. A particular focus in this system is on using templates to support the construction of measurement plan in the form of a set of goals, questions and metrics.

2.5.2.4 MVP-S

The MVP-S [Lot93], [LHR95] uses process notation called MVP-L for modeling processes, products and resources. The MVP-S execution environment supports the instantiation and execution of these models for the purpose of analysis, simulation, and project guidance. A primary goal of the MVP-S project is the explicit representation, collection, and use of measurement data by various editors. Such measurement activities are consolidated into the project plan by the project plan editor to describe the environment's work processes. During enactment by a Process Engine, there are a set of measurement tools to collect and store measurement data in a project database. The goals for each process model are specially adapted for incorporating measurement data.

2.5.3 Discussion

The presentation above has clearly demonstrated the importance of software measurement to improve productivity and product quality in software project. Measurement is a helpful means to assess, control project progress as well as predict project behavior. In addition, the survey also illustrates that measurement and modeling are complementary approaches which enjoy a synergy effect when applied together [LHR95]. A presentation of some measurement-based PSEEs demonstrates the feasibility of and benefit from integrating measurement capability to PSEE and reusing measurement data to better understand and effectively manage processes, e.g. selection of a more appropriate process alternative based on collected performance measures. A set of typical process and product measures are defined and embedded into the PSEEs. Such measures are then collected by tools which are automatically invoked. That is, the measurement activity (or policy) is incorporated into the process model. However, the study of such measurement-based PSEE also illustrates that collection of empirical data associated to software process evolution has received little or none attention. The measurement theory also identifies the three main entities to be measured in software process: process, product and resource. This observation is taken into consideration when we define the framework to characterize project in chapter 6.

2.6 Software Project Management

The high failure rate of large software projects in the 1960s had emerged increasing attention to software management. People began to realize that the problem lay in the management techniques used in carrying projects.
Roman in [Rom86b] states that projects are intended to produce certain specified results at a particular point in time and within an established budget. They cut across organization lines. They are endeavors, not completely repetitions of any previous effort.

Fayol in [Fay49] states that to manage is to forecast and plan, to organize, to command, to coordinate and to control. In [Loc93], Lock defines that project management is to plan, coordinate and control the complex and diverse of modern industrial and commercial projects. He also states that projects are more difficult to manage than other organizational activities due to their "one-of-a-kind" and multi-discipline nature.

Sommerville in [Som92] has identified following management activities as crucial: Proposal writing, involving a detailed specification of project constraints; Project costing, setting out an outline of the project work, cost and schedule estimates; Project planning and scheduling; Project monitoring and reviews; Personnel selection and evaluation; and Report writing and presentations. In the following sections, we will discuss some of those mentioned management activities.

### 2.6.1 Software Cost Estimation

Estimation of project cost and duration is a most critical task in project management. The estimation problem is inherent because we are trying to predict something in the future based on vague or evolving goals. On the other hand, project estimates are essential before the project starts, and we do not have other choices rather than doing our best to predict the project behavior in the future. Unless the project being scheduled is similar to a previous project, previous estimates are not a good basis for new project scheduling. Different projects use different programming languages and methodologies, which complicates the task of schedule estimation [Som92].

Boehm in [Boe81] discusses seven different techniques of software cost estimation:

1. **Algorithmic cost modeling** A model is developed using historical cost information which relates some software metric (usually its size) to the project cost. An estimate is made of that metric and the model predicts the effort required.

2. **Expert judgment** Estimates are determined based on knowledge and experience of persons who are supposed to have deep insight into the application domain. The final cost estimate is arrived at by consensus.

3. **Estimation by analogy** This technique is applicable when other projects in the same application domain have been completed. The cost of a new project is estimated by analogy with these completed projects.

4. **Parkinson's law** The cost is determined by available resources rather than by objective assessment.

5. **Pricing to win** The software cost is estimated to be whatever the customer has available to spend on the project. That is, the estimated effort depends on the customer's budget and not on the software functionality.

6. **Top-down estimation** The overall functionality of the system is examined, and cost estimates are made on the basis of the logical function rather than the components implementing that function.
7. **Bottom-up estimation**  The cost of each component is estimated. All these cost are added to produce a final cost estimate for the entire system.

Each technique has advantages and disadvantages. Several techniques should therefore be used in parallel. The idea of predicting costs from historical data and observed statistical relationships is known as *empirical cost projection* [DeM82]. This technique is identified as most prevalent in predicting future results. Cost projection requires some *quantum indicator* of the proposed project and an empirical database. This technique is based on the algorithmic estimation and estimation by analogy above.

The quantum indicator is required to be available early in the project. The approach of DeMarco to derive quantum indicators directly from the system specification, e.g. from data-flow diagrams [DeM82]. The approach of Boehm (COCOMO [Boe81]) provides a qualified guess on lines-of-code based on an algorithmic model relating to three software project classes (i.e. organic model, semi-detached mode, and embedded mode), and then use this as the quantum indicator. The approach of Albrecht (Function Point [AJ]) is to count “functionality” in terms of external inputs/output, user interactions, external interfaces and files used by the system.

The estimator is required to have two fundamental properties to provide accurate estimates.

1. **Domain experience** The estimator should have sufficient knowledge about past projects, preferably similar to the current one. In addition, he/she should also have knowledge about the application domain, customer, organization culture, tools and methods to be used, etc ....

2. **Estimation experience** The estimator has been estimating several projects, and his/her past estimates clearly demonstrate the appropriate competence. He/she is more likely to know the relevant factors (cost drivers) and their relative importance to project actual cost and schedule.

### 2.6.2 Scheduling and Resource Allocation

Project scheduling involves separating the total work involved in a project into separate tasks and assessing when these tasks will be completed. That is, time is specified to activities before project start and are continuously revised during project lifetime. The concept of time is essential to scheduling, often denoted as calendars. The output from the schedule process is usually a set of charts showing the work breakdown, task dependencies and staff allocation. Bar charts and activity network are graphical notations which are used in project schedule.

Resource allocation means to specify how people are allocated to activities which are not active. The activities should be designed independent of available resources. Several resource allocation scenarios can therefore be applied to the same activity network. Allocation and de-allocation of resources should only affect the duration of activities and not the network topology or layout. Priority rules are often adopted to different activities which compete for the same limited resource. An initial project plan is elaborated when all activities are given the resources needed.

The techniques in cost estimation is used to schedule software projects. Sommerville in [Som92] encourage project managers not to assume that every project stage will be problem-free. That is, estimates are first made as if nothing will go wrong, and then increased to cover anticipated problems by adding a *contingency factor* depending on project type, process parameters (deadline, standards, ...), and skill/experience of project participants.
2.6.3 Project Monitoring and Review

The objective of project monitoring is to ensure that the project stays in control or in planned schedule. DeMarco states in [DeM82] that staying in control means making sure that results make up to expectations. First, both projects and the surrounding organization should have reasonable expectations, i.e. the goal of project estimation and prediction. Second, we should manage our projects so that performance stays at or above some accepted standard. DeMarco states that

"You control a project to the extent that you manage to ensure the minimum of surprises along the way. The best controlled project is not necessarily the one that does the best or most work, but the one that best lives up to its predications. When there are deviations from what the project originally proposed to deliver, those deviations are minor and they are signaled early."

Project control is often synonymous to controlling cost and time; with an implicit assumption that quality products are produced. A project control system is used to measure project progress. The normal operation of many project control systems is to measure effort with respect to activities in a PERT chart. Each activity has an estimated effort, as well as planned start and finish dates.

Project rarely follow the initial plan. Thus, there is a need for re-planning during the project. It means that cost and time for upcoming activities must be continually revised reflecting the unexpected and unanticipated circumstances. The most frequent operations to revise an existing project plan are: adding new activities, removing upcoming activities, adding and reducing estimated effort for upcoming activities. Other operations such as merging, splitting of activities and moving estimated effort can be solved by a sequence of these basic operations above.

2.6.4 Discussion

Several commercial project management tools are available such as Microsoft Project, SuperProject. They assist managers in making initial project plan and schedule (often graphically represented as bar charts or activity network). They provide good recording facilities for estimates and actual values, but they do not provide support for creating estimates. Several alternative project schedules or resource allocation scenarios can be investigated before a final project plan is determined. During execution, the reported effort of activities are accumulated and presented by those tools to help managers to monitor or track project progress. On the basis of such progress information, the managers can eventually make appropriate decisions to revise the existing plan by carrying out re-planning operations (e.g. adding or removing activities; revising estimated effort; replacing or re-allocating scheduled resources; and so on) described above. After such re-planning operations, the existing project plan is adjusted to reflect the new state.

Despite of numerous features, the project management tools seem to behave as a stand-alone management support system. The exploitation of process knowledge and available past experiences has not been considered seriously. As a consequence, the plan is often heuristically made at project start and manually revised during execution in an intuitive manner. As software projects evolve rapidly during execution, the existing project management tools must constantly re-plan to adapt to changing environment without any adequate guidance and support. On the contrary, a PSEE offers features to describe processes formally in order to facilitate planning and enactment without having effective management support such as (re)-planning, tracking, etc. It means that the project management and process management disciplines can effectively gain mutual benefit from each other, and thus should be integrated
into a single environment. By doing so, the project plan become operational rather than being just a
document.

The contingency factor suggested by Sommerville (see above) is an effective, but naive approach to
anticipate problem in project scheduling unless the approach systematically reuses the past experience.
To accurately determine contingency factor, it is necessary to identify driving factors which cause
project turbulence and their consequences from previous projects. Such projects should be character-
ized according to properties that most likely influence required effort and schedule within a given type
of organization. Project properties have different impacts on project outcomes, and should therefore be
distinguished somehow. The COCOMO of Boehm characterizes software projects into three classes:
organic model, semi-detached mode, and embedded mode. Such project classification is coarse-grained
and highly qualitative to determine similarities between projects. Historical data on estimated and ac-
tual effort/performance should be recorded and associated with project characteristics. When a new
project is planned the estimator will look for similar projects and use this as a basis for the making
estimates.

2.7 Examples of Software Experience Databases and their Usages

In this section, we briefly present some examples of real software experience databases and how they
are used to support software development and maintenance. This survey also aims to illustrate the
importance of and benefit from empirical study in software engineering.

2.7.1 Experience Database at NASA-SEL

[MPP+94, BG94] describe an improvement program which lasted from 1976-1991 where over 100
software development projects were studied. Their cost, product and process data were collected and
used to build typical project profiles, against which ongoing projects can be compared and evaluated.
Two experiments described in [BCM+92] was performed at NASA-SEL to introduce Cleanroom and
Ada/OOD to pilot projects. The results obtained were compared with the projects using standard SEL
methodology. Over 6 years of experiments and continual refinement the new technologies are stabi-
lized and installed among SEL standards. However, it took further 3 years to establish a baseline for
projects using new technologies.

The indicators for improvement on software process comparing with status by the end of 80’s by ap-
plying SEL Process Improvement Paradigm are:

- The cost per line of new code has decreased slightly about 10% when the increase of problem
  complexity and improved functionalities have not been taken into consideration.
- Higher degree of reuse of code, design, methods, test data, etc...) makes the overall cost for
delivered system to reduce considerably.
- Reliability of the delivered system has improved by 35% along to the fact that number of error
  per KSLOC has reduced from 8.4 to 5.3
- Software process is more manageable in the sense that project managers can better plan, predict,
  control the cost and quality of product being developed based on SEL’s various models (error
characteristics, change/defect, product, process, quality, etc...), well-defined methods. Extreme uncertainty in estimating such as cost, staffing, size and reliability is about to be disappeared.

- Effort spent in rework (i.e., defect detection and removal) is also decreased.

Total effort for process improvement activity is 11% of total expenditures. However, the pay-off or return of invested effort has not been derived yet.

The Software Management Environment (SME) was developed in the Software Engineering Laboratory (SEL) of the NASA Goddard Space Flight Center. This system was developed primarily for use by managers and greatly simplifies the work of comparing collected data with baselines. Baselines are derived using data and experiences gathered from prior projects.

### 2.7.2 Experience data at Software Productivity Research (SPR)

Capers Jones has reported his observation of thousands projects within hundreds of organizations in [Jon96]. The failure or cancelation rate of large software system is over 20 percent. Of the large system that are completed, about two-third (e.g. 54 percent) experience schedule delays and cost overruns that may approach 100 percent. About the same number of large systems are plagued by low reliability and quality problems in the first year of deployment. However he claims that successful projects all tend to follow similar patterns of development practices regardless of what industry domain or country develops them. Based on the project data, he derives a relation between deployed technologies and observed social factors on one hand and unsuccessful-successful projects on the other hand. He has also investigated the quantitative consequence of system size to project deliverability (classified by early, on-time, delayed or canceled).

### 2.7.3 Discussion

From the above presentation of existing software experience databases, it clearly illustrates the fact that we can benefit from collecting and learning from the past project experiences for future projection. However, there unfortunately exists no global experience database from which all software organizations independent on size, application complexity, industry domain, culture, etc., can effectively benefit. That is, the experience gathered by one organization is not necessarily valid or applicable for other organizations because of their inherent differences in the context, complexity, application domain, countries, surrounding environment, and so on. A technology which is persistently proven to be suitable for one software company, does not necessarily conform to another company as well. Experiences must therefore be locally collected and established by a software organization itself. The collection and establishment of experience data is also shown to be time-consuming and costly before the data foundation is considered as valid and reliable. Therefore, it is important to have clearly defined objectives of the data usage and full commitment.

### 2.8 Evaluation and Open Issues

In the following, we summarize the evaluation of existing technologies from our discussions above followed by an outline of relevant issues which deserve to be investigated closer. The identified open
issues serve as a starting point for the next chapter in which requirements are stated to found basis for our work directions in the remainder of the thesis. Those identified open issues are also evaluated based on our research achievements in section 10.5.

PSEE

The assessment of PSEEs in section 2.2.3 reveals the fact that none of PSEEs independent on the underlying modeling paradigm of the language, *fully* satisfies the need for software process management e.g. explicit representation of process components (activity, product, organization, tools, etc); measurement support; project management (planning, scheduling, tracking); version/configuration management; transaction and workspace to handle concurrent engineering. .... We therefore perceive a need for taking a step backward to identify the particular objectives and requirements to the research on software process and specifically process support in a PSEE. On that basis, we can seek appropriate techniques from existing PSEEs or from other disciplines to be adopted in our PSEE, namely EPOS. Open issues which need to be investigated closer, can be summarized as follows:

- **Tight integration** of configuration management (CM) and process management (PM) in a single PSEE.
- **Flexible means** for defining project measures to be collected and analyzed.
- **An embedded experience database in a PSEE** to record project history, including project characteristics, estimates, actual performance, estimation capability, recognized evolution patterns and evolution data.
- **Operational support** for both top-down and bottom-up process model evolution.

Software Evolution

Lehman’s work emphasizes the importance and significance of classification to better understand of and predict future evolution of software systems. Evolution of software applications during both development and maintenance process is the main focus of this work. On the contrary, Madhavji’s work provides a comprehensive model for both i) describing software processes (including basic items of change i.e. policy, law, people, process, resource and result) together with their interdependencies and ii) classifying and recording changes for future projection. The two models are of great prominence and learning value within research on software evolution. However, they are still defined at a methodological rather than an operational approach. No tool supports have been given, nor attempt to integrate into a PSEE exists.

When discussing the linkage between process and its model, Lehman in [Leh94] states that there is a need to adapt process to specific conditions or unanticipated circumstances, and model evolution is basically a consequence of process evolution. Most PSEEs are concerned to solve the problem of changing process models and then migrating new changes to process model instances (cf. our assessment in section 2.2.3). Such changes are considered as *planned* process evolution. Often unanticipated circumstances and unexpected conditions lead to process instance adjustments, adaptations and changes on the fly [Leh94]. Such *unplanned* process evolution which occurs more frequently than *planned* one, is not necessarily incorporated to the process models, but need to be anticipated and thus effectively
managed. Based on the discussion in section 2.4.5, we identify following open issues needed to be treated in our work and are evaluated in section 10.5.

- It is necessary to have a more flexible and configurable framework to categorize process evolution specifically the unplanned one. This will lead to increasing understanding of the evolution properties and impacts to software project.

- It is important to define methods or approaches to manage such process evolution should be operational and integrated with a PSEE.

Project management

The discussion in 2.6.4 illustrates the need for integration project management facilities, particularly re-planning operations, in dealing with unplanned evolution during execution into a PSEE. On one hand, the project management technologies can benefit from PSEE’s capability in modeling, planning, enacting and evolving process knowledge or models. On the other hand, the process technologies in PSEEs can take advantage of tracking and re-planning (i.e. adding/removing activities; adding/reducing estimated effort, re-allocating scheduled resources) facilities offered by project management discipline. It is important to realize that research fields are strongly related. We should always keep track of work in related areas, looking for opportunities of cross-fertilization and mutual benefit. The discussion results in following open issue needed to be examined closely:

Coarse-grained process follow-up and poor product modeling  Project management tools provide process follow-up in software engineering projects in a fairly coarse-level, often by a project plan through PERT charts. The emphasis is on effort/time registration against milestones and on testing and error recording according to a quality plan. There is little explicit product modeling.

Lack of embedded and global experience database  The assembled project data is often paper- or spreadsheet-based, i.e. not commonly available in an online and central experience database for reuse across projects in the organization.

Poorly managing unpredictable process evolution  The conventional project management tools are hampered by massive and unpredictable process evolution during project execution, e.g. because of unstable customer requirements. The integration of project management facilities into PSEE will give mutual benefits for better planning, scheduling, monitoring and thus adjustment software project.

Software measurement and experience reuse

The discussions in sections 2.5.3 and 2.7.3 on software measurement and project experience respectively demonstrate the importance of recording relevant project measures to direct and predict the behavior of new future projects. Software engineering has been criticized from being too much advocacy research rather than based on empirical studies. By effectively exploiting technologies in software measurement and experience reuse, we are able to get more quantitative and statistical control over the process, and thus the product quality. It is also crucial to integrate those available technologies into a PSEE to better manage the evolutionary nature of software environment. In particular, we
aim to utilize measurement to best characterize software projects, and lessons from project experience reuse (e.g. NASA-SEL, Basili’s Experience base) to effectively anticipate process evolution.

2.9 Chapter Summary

In this chapter, we have presented the state of the art of software engineering and its related topics. We identify six relevant issues for the thesis. The chosen topics are:

1. Software engineering and its associated processes. We discuss the software engineering discipline in general followed by the presentation of various life cycle models. We then discuss the effort in improving the general life cycle models and in introducing the integrated environment to describe process and support process performance (PSEE). Some typical PSEE are briefly described followed by an assessment to identify strengths as well as weaknesses of existing PSEE.

2. Software process improvement models. We discuss five models which provide systematic approach to improve software processes. They are ISO 9000 series, SEI’s CMM, QIP, BOOTSTRAP and SPICE. A discussion on their application to software organization is briefly described together with a tabular comparison of those models.

3. Software evolution. This is probably first termed by Lehman and Belady when investigating the dynamic and evolutionary nature of software system. We present two prominent evolution models of Lehman and Madhavji. The former focuses on the item of change, while the latter concern with the classification of change and its properties.

4. Software measurement. It is widely recognized that we need measurement to control software project. DeMarco states in [DeM82] that “You can’t control if you can’t measure”.

5. Software project management. We investigate the features offered by software project management discipline to seek for possibility in integrating with PSEE.

6. Software experience database. Three examples of software experience databases are described to illustrate the usefulness and benefit from past project experience in performing new future projects.

A more or less comprehensive discussions for all six topics are given and summarized in section 2.8. A set of open issues are then identified directing our further research focus. Such identified work issues will be detailed in the next section which states a set of requirements for our PSEE.
Chapter 3

Objectives and Requirements

3.1 Introduction

In this chapter, we present the general objectives of research on software process followed by a discussion of relevant issues to be addressed. Based on the discussion of research objectives and issues of software process, a set of concrete requirements is then stated for process support in a PSEE dealing with different aspects, e.g. modeling concepts, modeling language, instantiation and enactment of process models, data management, cooperation, organizational modeling, project management and support for evolution. Those requirements have guided our research directions during the last few years, and thus laid a foundation for the remainder of the thesis. The stated requirements are also used to evaluate our achievements reported in subsequent chapters. Requirements intentionally elaborate the issues which are identified in section 2.8, and are divided into three main areas of concerns and structured as following. A basic set of requirements for a PSEE will be outlined in section 3.6. Additional requirements supporting software engineering project are described in section 3.7. Section 3.8 elaborates in details the motivation, needs for managing process evolution. First of all, we give in section 3.2 a conceptual reference framework which is used in the rest of the thesis.

3.2 Terminology Clarification

It has been over a long time recognized a terminological problems within software process community. A lots of effort have spent in defining a common terminological framework for software process such as [CFFS92, FH93, Lon93, CFF94]. Although they have laid a solid foundation for common understanding and comparing different concepts, the terminological problems don't seem to be completely eliminated. People keeps using different concepts referring to the same meaning and vice versa. Therefore, we need to clarify a consensus on how different concepts are meant and consistently adopted in this thesis. Our adopted terminologies are very much inspired by the above reference frameworks for process concepts.

We distinguish between a changing real world where managerial and technical activities are taking place, and a modeled world where the human perceptions and constraints of the real world is represented by (preferably) formal models and documentation. The former is continuously evolving due to revised requirements, changing platforms, improved understanding/insight, etc. The latter remains static, until humans determine to change it according to the world it reflects. Thus, the modeled and idealized world is more manageable, and its evolution is more easily monitored and controlled. Figure
3.1 depicts the process concepts, and their interactions.

![Diagram showing process concepts and their interactions]  

**Figure 3.1: Software Process and their models in a PSEE**

In the following, we give a more specific explanation of the terminologies in figure 3.1 above.

**Activity** is a step of a process producing externally visible state changes to the software product. It incorporates and implements rules, strategy and procedures and aims at generating or modifying a given set of products. Activities may be organized into networks with both horizontal (chaining) and vertical (decomposition) dimensions. There may be both production activities and meta-activities. Activities which are associated with roles, tools and products, span all the different categories of aspects in the software process, e.g. managerial activities like planning, monitoring and follow-up; cooperative activities like reviews and inspections; and engineering activities like coding, testing, etc.

**Product** is an artifact which is produced by performing an activity. The same artifact may serve as input to another activity to produce other products. For the production process, the products are e.g. requirement documents, source code, executables, user documentation, etc. They should be persistent and versioned. Product persistence facilitates sharing, history tracking and analysis, while they are versioned due to the high rate of changes in software process.

**Production tool** is computer program supporting or automating a part of the work related to an activity. Tools are characterized by the operations they implement, their cost of use, and their availability.

**Project** is a specific instantiation of a process to produce a specific product in a given organization, with specific objectives and constraints (e.g. budgetary, schedule, available resources, tools, staff,
etc.). The project is usually staffed by several people having different skills and expertise. It is thus important to effectively coordinate the required work among available resources to achieve optimal result. In addition, working in team requires an effective communication aiming to synchronize intermediate works into a common goal. The generated products should be effectively shared between project staff to optimize efficiency and productivity (e.g. collaboration). The aspects of coordination, communication and collaboration are governed by a set of rules, policies or procedures - e.g. cooperation protocol or agreement.

**Organization** is a rigid, pre-established command line and decision making authority unit in which agents (human or specific persons) are performing activities related to a role. A role describes a set of responsibilities, rights, and skills necessary to accomplish a specific activity in the software process. Relevant issues to be taken into consideration in this area are: e.g. organizational structure (matrix, hierarchical, ...); groups/sub-groups; roles, responsibilities and people.

**Production process** can be coarsely characterized as a partially ordered network of interacting activities, aiming at developing and maintaining the product to be delivered. It encompasses the areas discussed above: activities, products, production tools, project and organization.

**Meta-process** is a set of partially ordered activities which maintain and evolve (modify, refine, customize) the whole process, (i.e. production process, meta-process and process support) in order to cope with different change requests. Such change requests can possibly originate from internal or exogenous sources.

**Process modeling language** (PML) is a formal notation used to express process models, both for production processes and meta-processes.

**Process model** is a description of a process expressed in a suitable process modeling language (PML). A model is always an abstraction of the reality it represents, and is as such only a partial description. That is, there are always parts or aspects of the process, that are not captured in the model. The process model which often consists of types and instances of these, can constitute several sub-model representing production process and meta-process. Some examples of production process models are: activity model, product model, tool model, organization model and cooperation model. Those models will reflect their relevant corresponding aspects described above. The real-world processes and their corresponding models should preferably be kept synchronized. A process model is composed of several model variations, holding refined or customized embodiments and representations of the same model. Variations can be classified into three different categories or levels:

**Generic** or template model variation: is the definition of the process model as expressed in the PML. Depending on the underlying paradigm of the language, a template may consist of type definitions, re-entrant code, or simply source code that may generate several run-time occurrences.

**Enactable** model variation: is a uniquely identifiable instance obtained from a generic variation. It contains the necessary context-specific information to make it possible to execute it. It is thus possible to have several enactable variations originated from the same generic variation.

**Enacting** model variation: is created when enactment actually starts, with a binding to a process model interpreter and with an explicit enactment state. An enacting variation is created from an enactable one.
**Process support** consists of process models, process modeling methods and language (PML) and process tools. The process tools manipulate (create, modify and evolve) process models. A PSEE is an environment that provides such process support.

To summarize, the production process and meta-process are entities of the real-world processes which are represented by the internal, computerized process models to enable support and control. The process models which are expressed in PML, are built, modified and evolved by process tools according to the evolution of the real-world.

### 3.3 Objectives of Research on Software Process

The software process has in recent years been recognized as crucial to improve various attributes such as productivity, product quality, time-to-market (or lead-time), and predictability of software production [LB85, Hum89]. Such recognition has been proven by the increasing research effort spent in and focused on more adequate concepts and formalisms (e.g. object-orientation, specification languages, fourth generation language); development methods (e.g. waterfall, fast prototyping, reuse); and software process improvements (CMM, ISO series, QIP, etc) over the last decades. However, the exact influence of the software process in improving the above mentioned attributes is still not well understood. This is due to e.g. lack of systematic and empirical studies of software processes. In the following, we will discuss the general objectives of research on software process as a motivation to identify specific issues which need to be addressed properly in a PSEE.

#### 3.3.1 General Research Objectives/Directions

There exist a numerous of objectives/goals with respect to research on software process depending on what kind of support people wants to seek. We have identified some major objectives or research directions. Note that the list is not exhaustive.

**Process assessment** The research wants to investigate the actual software processes by empirical study to reveal undesirable or deficient process properties. The assessment can be carried out based on a predefined standard of best practices. Depending on the assessment results, effort is spent to find alternatives to eliminate the undesirable properties of processes - presumably contributing to improved processes.

**Process simulation** The primary focus of this research is to gain thorough analysis of an imaginary software process without the cost of actually executing it. Performing simulations (or dry-run) on the imaginary software process, rather than the actual one, is thus considered as a cost-effective, low-risk means to identify undesirable features (cf. the above mentioned process assessment operates on the actual process). People can try out some hypotheses by simulation before they are eventually incorporated into the real, improved process.

**Process learning** The research aims to gain better understanding of and insight into software processes. The gained process knowledge is then disseminated to promote process awareness and maturity among software practitioners. By doing so, it presumes that people will work better, more efficient in the process. This research yields various study in e.g. knowledge discovery, representation and modeling.
3.3. Objectives of Research on Software Process

**Process support** The research aims at providing computer-based support for performing software process within an integrated environment. The main objective is to provide a software engineering environment (e.g., PSEE) in which processes are consistently enforced to effectively support and guide software activities. Research on process support consists aspects of process modeling, enactment and management.

It is worthwhile to note that those research objectives above are to a certain extent related and complementary. E.g. process assessment, simulation and learning can easily be performed in the context of a process support in PSEE. Therefore, we will in the next section study in details the concrete objectives of process support in a PSEE.

### 3.3.2 Objectives of Process Support in PSEE

The overall objective of the process support in software engineering is to deliver high quality product, on time, with reasonable cost and within scheduled budget. The rationale of process support can be viewed from different perspectives: e.g. customer, management and engineers. The customer basically perceives process support as a means to achieve a full product satisfaction in accordance to their requirements and needs. The management is interested in increasing competitiveness, and reputation as a reliable software vendor. The individuals want to have a meaningful, motivated and personal job satisfaction by fully understanding of and flexibly accommodating their own processes. Below, we outline a set of main objectives of or expectations to process support in a PSEE.

**Improved productivity** It means that a required software product can be developed in fewer work hours, with less resources. For the management, it implies reduction of development cost, and thus an increase in profit and competitiveness. The customer can also get the required product at lower cost.

**Improved product quality** The developed product is well complied to the specification, or to customer expectation. High quality products result in good reputation, increasing sales/market share and reduction of maintenance cost for the company. For the customer, high quality product reduces the cost of operation, support, and increases end-user satisfaction.

**Decreased time-to-market** It means that the required product can be developed and released to market in a shorter calendar time. Such reduction of time-to-marked or lead-time strengthens the company’s competitive advantages by bringing new product and technology to the market sooner than others.

**Better control** The process support technology expects to improve ability to predict performance of production process (i.e. on time, within budget). By incorporating software measurement (measures, metrics) into process support, the production process can be brought under statistical control. E.g. the time to release a product can be determined based on a quantitative foundation with high confidence. However, the software engineering discipline is characterized as experimental and evolving - The absolute control over software process is therefore hardly obtainable.

**Improved process awareness** The individuals have full understanding of and insight into their processes, also including rational, purposes and goals. Such a process awareness is crucial for motivation and thus engagement in detecting and then improving undesirable process features. The
process support provides means to explicitly define processes to be followed, and goals to be aimed at. In that way, the individuals can easily compare and evaluate their performance against goals to get a sense of progress and accomplishment.

Better conformance It implies that the individuals are enforced to conform to the defined process description in their works. By consistently following the defined steps, the individuals will likely develop best quality product within scheduled amount of resource. However, the process support must also facilitate automation of the trivial tasks, and eliminate human concern freeing up time and energy for the more creative tasks.

3.4 Our Underlying Research Objectives

Based on the general objectives above and our motivations for research in section 1.2, we state below a set of our research objectives. Those objectives which are an elaboration of identified motivations, will be evaluated in section 10.4.2.

O1. Integrating SCM and project management into PSEE There has been a large amount of effort dedicating to research on software process technology. A large number of PSEE with focus on process management (PM) which have been developed in the last decade - some are developed as research prototypes while some are sufficiently mature to be commercial. Already in early of 1980s people began to research on software configuration management. Still, the two research directions (SCM and PM) seemed to be divergent rather than convergent and coordinated. Such divergent development of software process technology was largely common in most PSEE in the late 1980s and early 1990s. Few or none of the existing PSEE offers a satisfactory amount of functionalities within an integrated environment including e.g. configuration management, process management and project support (see more in chapter 2). Our motivation or rather vision is therefore to provide such a complete, excellent and integrated PSEE by thoroughly analyzing "best practices" from other PSEE and then directing our research focus. Our PSEE will provide us a platform to experiment and validate our new research ideas.

O2. Managing process evolution within PSEE context It is necessary to define a flexible framework to categorize and thus better understand the nature and properties of unplanned evolution. Such categorization framework can be used to systematically record evolution occurrences, and then recognize most frequent evolution patterns together with their associated impacts. The details of the framework should be flexibly customized to adapt to specific needs of a particular organization.

Software project is best characterized as experimental science, i.e., that each project has an unique goal, specific assumptions and constraints. No two projects are alike, neither are two evolution occurrences. It is therefore important to relate recorded evolution data to the project context in which they are captured. Existing technology in software measurement should be taken into consideration to define a minimal set of measures which best characterize projects. If evolution still occurs during project execution, it is important to have adequate, iterative support to adapt to it and then predict its consequences/impacts. The project plan must thus be adjusted accordingly. We have observed that many existing project management tools provide suitable supports to dynamically revise the project plan. Such re-planning features should be investigated and incorporated in a PSEE.
3.5. General Research Issues

Learning from failures or successes of past experiences is innate for human being, and lessons learned from past projects are thus not of exceptions. It has been proved in literature that there exists a consistent relation or correlation between a given project characteristics and project performance. Many have also stated that it is hard to provide statistically significant evidences on such relations. However, we believe that the same evolution patterns will very likely occur again and again in similar projects, and this claim can only be confirmed or refuted by conducting empirical studies with real software projects. The reuse of previous experience on evolution will help to make the original project plan better prepared for changes. That is, the impacts of recognized evolution patterns are anticipated in the original plan before project starts.

O3. Validating PSEE  Concrete benefits of PSEEs for software practitioners have not been properly validated by industrial case studies or experiments, neither for process modeling nor for process enactment. Our objective is to properly validate our PSEE prototype which has integrated the SCM, project management facilities and process evolution management. The validation work should be seen as two phases:

1. Internal validation with process examples to try out our modeling concepts and enactment facilities.

2. External validation with real industrial setting mainly to assess the requirement for formal modeling language and for computer-assisted process enactment. In addition, the industrial case study will provide us real data on process evolution and opportunity to validate our framework and approaches.

In general, we are basically addressing the issue of unplanned evolution and finding how to better manage it by combining existing technologies in different disciplines within an integrated PSEE context. We can benefit from e.g. other PSEEs, SPI methods, existing research on software evolution; software measurement, software project management and reuse of project experiences. We believe that ability to anticipate unplanned evolution and effectively react to them is a characteristic feature for project success. Because the occurrence frequency or arrival rate of the unplanned evolution is much higher than of the planned one in most software companies, the research in this thesis is of great value. By addressing the issues mentioned above, the evolution is probably better predicted and controlled. That is, the unplanned process evolution is better managed within the context of a PSEE.

3.5  General Research Issues

As the presentation of research objectives is given in section above, we will now in this section outline some relevant research issues which are related to software process. Those issues will be elaborated when we state more concrete requirements to a PSEE in subsequent sections. We apply terminologies as they are described in section 3.2.

3.5.1  Overall Issues

The following issues concern to the software process as a whole.
Process concepts To understand, communicate and eventually model processes, we need to establish a set of process concepts, related terminology, and relationships between them, i.e. process framework or conceptual model as described in section 3.2. As also mentioned in that section, steps towards standardization of concepts and terminology are important in increasing a common understanding in the process community. Such standards can not be established over the night and then gained common acceptance. Only through an iterative process of definition, use and evolution can such standards of concepts can be fully developed.

Empirical study The software process research is criticized to be highly laboratory, and lack of empirical studies of real, industrial processes. This has surely widen the gap between software scientists and practitioners, when concrete and tangible evidences on benefits fail to appear. Lack of empirical studies in real process contributes to loss of fidelity and thus financial support to our research. Also, we need to examine, model and assess actual, industrial processes in order to clearly demonstrate the benefits of process support in making software process better.

Human and social aspect Humans and their social aspects are major factors in software processes. It implies that humans, their behavior and perceptions must be, to a rational extent, described in process models. As human behavior is complex, unpredictable, and human perceptions are vastly differently from one person to another, it is really a tough issue for process modeling. Modeling of human aspect must therefore be restricted to such an extent that it allows certain degree of freedom and flexibility for the individuals. Comprehensive models for human behavior is beyond of our scope. Such issues are treated by other discipline to a much larger degree such as psychology and social sciences.

3.5.2 Production Process

Below, we outline relevant issues relating to production process which as depicted in figure 3.1 includes activity, product, production tool, organization, project, etc.

Influence of process It is necessary to examine the impact of good process on quality of developed product and productivity. Such knowledge can only be retrieved by conducting several controlled experiments to collect relevant empirical data on process impacts. There exist of course a large number of dependent process variables and thus make the controlled experiments hard and complex. However, it is significant for software research to establish a quantitative relation between software process and its other attributes (e.g. product quality, productivity, cost, time-to-market, ...).

Product variability Software products can be characterized by a large range of attributes, e.g. size, complexity, domain, application type, lifetime, architecture, etc. Such product attributes should be taken into consideration when the production process is defined or configured. As an example, a large-scaled software system has stronger requirements for formal procedures than a small one does.

Tool variability The production process should be configured to be independent with the particular tool set to be used. In that way, the defined processes are more flexible and susceptible to environmental changes.
3.5.3 Meta-process

The meta-process is in charged of defining, customizing, and managing the entire software process, including production process, process support and meta-process itself. Process customization is adjustment to particular circumstances while process management implies tracking, monitoring, assessing and improving process. The production process can be improved by e.g. infusing presumably better method, technique, or tool to replace the existing ones. The process models, process modeling language and associated process tools can be improved according to rules, guidelines or procedures defined in the meta-process. We identify some relevant research issues below relating to meta-process.

Customization The process is initially described in a very generic manner giving space for flexibly configuring according to specific circumstances (e.g. specific project constraints). The meta-process should hold rules, guidelines to describe circumstances and criteria for selection between anticipated process variations as well as extraordinary circumstances where human judgment must be involved.

Planned or unplanned change We distinguish between meta-process support for planned and unplanned changes. Planned changes are mostly initiated based on a need to improve or move to a better state. On the contrary, unplanned changes occur unexpectedly and often cause great extent of impacts. The meta-process should be defined to deal with both types of changes both before and during process execution.

Transition of change It concerns with implementation or installation of changes to real-world processes. It can e.g. include installing a new tool, updating process manuals, rewriting guidelines or revising data collection forms, etc. Such activities should be explicitly specified to ensure that the new changes are actually implemented both at the process- and the model-level (see also the synchronization issue of process support below).

Recursive problem The meta-process is also subjected to change, i.e. the meta-process changes itself. This implies to the recursive problem as the meta-meta-process will describe how the meta-process is changed. The issue is where to stop to make a transitive closure of all levels of meta-processes and support. Reflective architecture in which a system can reason with and modify its own representation, is a possible solution.

3.5.4 Process Support

As mentioned above, process support consists of process models, process modeling language and associated tools which are used to manage (modeling, enacting or evolving) the two former. There are some research issues needed to be considered concerning to process support.

Synchronization This issue concerns how to keep process models (i.e. computerized process representation) and their reflecting real-world processes in synchrony. That is, it is necessary to synchronize the internal process models (i.e. what is expected to be executed) and the reality (i.e. what is actually executed). The synchronization feature is a vital prerequisite for that a process support environment is performing as intended, i.e. assisting and guiding software processes.

Modeling intention There exist three manner to model software processes. Prescriptive models make explicit what should be done or happen similar to a cookbook recipe. Proscriptive models make
explicit what should not be done or happen. A descriptive model is orthogonal to pre- and pro-
scription. The descriptive statements are concerned with what actually is, i.e. facts of phenomenon.

**Flexibility or control** When enacting the process models, there are two modes of relevancy to reflect
the extent of flexibility and freedom. By process support, we are either provided with facilities
to flexibly configure the intended process, or enforced to conform to a set of predefined rules.
Strictly enforced control easily leads to the so-called strait jacket effect, and thus promotes non-
acceptance or dissatisfaction.

**Guidance or automation** Guidance represents a way of providing support in circumstances where
automation is hard or impossible. In that way, the user and the environment cooperate to achieve
a common goal. That is, the user makes decisions which can not be arrived at automatically. The
environment provides advice or guidelines, based on stored information and knowledge about
the activity and goals of the users. Automation implies a mode of activity in which the environ-
ment by itself, based on the occurrences of particular states, decides actions to be performed. In
fact, we need both guidance and automation in process support. The environment should be able
to offer advice and guidance, and at least preferably, that advice or guidance should be offered
automatically.

**Re-active or pro-active** These two terms are related to aspects/mechanisms of activation or enact-
ment. Re-active implies that the main responsibility for initiating activities, or more generally
for progression, is with the user, while the environment augments the user, e.g. as by control or
advice. Pro-active implies a shift in the responsibility for progression towards the environment.
Pro-activeness thus requires a much higher, and often hard to achieve, level of knowledge and
automation as an integral part of the environment.

**Evolution of process support** There are two entities which are subjected to change: process models,
and PML together with process tools. The process models are evolved as a consequence of or
to reflect changes done to the real-world processes (cf. synchronization issue above). On the
other hand, if process models are revised within a context of an improvement initiative, the new
changes must be transferred or implemented in the real-world processes (cf. transition of change
issue above). In both cases we should regard different requirements for evolution of process
models at different levels or variations. The PML and its associated tools may also be changed,
e.g. extension of PML constructs or semantics. If all existing features of PML and process tools
remain, there is no need to change the existing process models. Newly created process models
must of course conform to the extended features. In case features of PML and process tools are
updated, the existing process models must be modified accordingly. This is a complex task and
must be considered properly.

In following sections 3.6, 3.7 and 3.8, we will detail the research issues mentioned above, and then
state a set of concrete requirements for process support in a PSEE.

### 3.6 Basic Requirements for a PSEE

In this section, we present a set of basic requirements needed to be met by a PSEE. Those stated re-
quirements include various relevant aspects for a PSEE such as modeling concepts; process modeling
language (PML); process model construction and process model enactment.
3.6.1 Modeling Concepts

Modeling concepts include a collection of relevant process terminologies terminologies and their interactions (c.f. those described in section 3.2 above). Recognition or definition of necessary modeling concepts for constructing process models is a fundamental issue to be addressed by a PSEE at first. That is, the defined concepts must be able to describe the most interesting real-world phenomena in software development and maintenance processes. The set of defined modeling concepts must satisfy following requirements:

**Expressiveness** The semantics of defined concepts must sufficiently express aspects and domain of interest in software processes. However, it is impossible to quantitatively assess that a set of defined modeling concepts is sufficient. This property may be evaluated by examining its capability in expressing real software processes. There are several process aspects to be taken into considerations when elaborating the conceptual framework. These aspects concern e.g. software engineering activity, product, adopted production tool, organization, resource, ....

**Economy** The collection of the defined modeling concepts should be a minimal to be able to capture and represent process aspects necessary to satisfy specific needs of a PSEE. This requirement is however conflicting with the previous one. The degree of achievement of those requirements can thus be adjusted or tuned to meet a particular goal.

**Consistency** The defined modeling concepts are dependent to each other one way or another. Such inter-dependency should be consistently defined in such a manner that the modeling concepts appear to be well-structured and systematic. That is, neither redundancy nor logical inconsistencies should exist.

**Familiarity** Modeling concepts and their interactions must be able to reflect a similarity or resemblance with respect to perception of real software processes. It means that defined concepts should easily be recognized and associated to the understanding of the real world.

3.6.2 Process Modeling Language (PML)

As soon as the conceptual framework is elaborated, we can then think of presenting modeling concepts in a formal manner. Formality means to put syntax and semantic to the defined concepts which are used to build process models. The intention of having process models is to facilitate communication and human understanding of the complex real-world processes. The models can describe real-world processes in three distinct ways: i) *descriptive* - describing the process as it is; *prescriptive* - describing the process as it should be carried out; and *proscriptive* - describing the process that is forbidden to perform. In addition, formal process models facilitate machine interpretation, analysis, reasoning and execution to assist in process performance. The PML is required to meet following issues:

**Understandability** PML is meant to ease communication between people involved in a software processes. It implies that PML must be easily understood by any one with different skill and knowledge background. It can be done through an external graphical notations.

**Abstraction** The definition of PML should facilitate possibility for describing process models in different levels of details. That is, it allows process models to be hierarchically decomposed and
gradually specified. This feature is essential to hide irrelevant details of model information until necessary. This abstraction property also supports and improves the requirement on understandability above.

**Modularity** PML and its language constructs should facilitate encapsulation (grouping, information hiding) of process models. In addition, reuse of process model fragments should be done in a systematic manner. Modularity of process models thus facilitates maintainability and possibility for reuse. Process models should be organized in such an extent that their model fragments are easily redefined and reused in other context. This feature may lead to a search of a well-balanced ratio between coupling and cohesion of process models. In addition, other techniques, e.g. parameterization; body/interface; import/export; private/public; …can also satisfy this requirement.

**Evolution and Customization** Real-world processes are constantly subjected to change due to the dynamic environment they are operating in. It is well recognized that scientific knowledge acquisition is a gradual and progressing process. Information originally embedded in process models is usually of incomplete, uncertain and possibly generic nature. When improved insight is gained, both the structural and behavior aspects of process models must possibly be updated, extended or evolved to reflect the changes in the business and to absorb improvements. All the three process models (generic, enactable and enacting) must be continuously evolved. The definition of PML should be open-ended so that new modeling concepts or notations can easily be integrated and accommodated according to specific needs.

**Formalization** The PML and process models must be formal to enable automated analysis and assessment. The objective of the analysis is to detect syntactical flaw; conflicting conditions or goal; and actual or potential inconsistencies. This requirement is conflicting with the understandability requirement above. Fully formal process models are often hard to understand than process models which are expressed in natural language.

**Clarity and Orthogonality** PML should facilitate that a small set of well-defined concepts can be freely combined.

### 3.6.3 Process Model Instantiation

Process model instantiation means to create a enactable process model variation from a generic one. Such instantiation is necessary to construct an enactable process model which is possible to be executed, binding with actual process parameters or constraints (e.g. time, cost, resource, products). The instantiation or planning process should satisfy following requirements:

**Automation** It is required that the process instantiation process should be performed with minimal human intervention as possible. That is, there should exist mechanisms and algorithms intelligently reasoning about existing information and issued goal (computer-assisted intelligent automatic planning).

**Increment** The planning mechanism should gradually expand enactable process model variation when necessary. This requirement restricts the number of generated instances to the level in which user can stay in control. In addition, incremental planning better reflects new changes to the generic
process models. I.e. if dynamic binding of process model information is used, changes can be automatically incorporated into the newly created enactable process models.

**Ordering** The enactable process model should contain activities which are either vertically decomposed into several specific ones, or horizontally connected. In that way, activities can be executed in either parallel or sequential manner.

### 3.6.4 Process Model Enactment

The key objective of a PSEE is to provide adequate assistances in performing processes in a structured and standardized manner based on a set of described model. It means that there is a need for having adequate mechanism to interpret and execute the enactable process model variation. As the enactable process model is bound to a process interpretation engine, we have an enacting process model. However, it always raises a impetuous discussion when the term enactment is mentioned. It often claims that process enactment only implies strict enforcement and thus rigid control of process performance without human intervention to affect the sequence of execution. The pertinent question is whether process enactment actually provides strict control or flexible assistance. Our requirements stated below attempt to search for a balance between two extremes.

**Automatic execution** The enactment mechanism must provide facility for automatically executing trivial and time-consuming tasks. This feature give effective assistance and release unproductive tasks from process performers.

**Interactiveness** It is also important for enactment mechanism to interact with process performers and be pro-reactive to any response or changes. This requirement which preserves control for human, is meant to restrict the degree of automation stated above.

**Performance** It is efficient to store stable and invariant data structure and information for process enactment in a rapidly accessible data media. By doing this, we achieve faster response time and thus considerable improvement of performance efficiency.

**Portability** The enactment mechanism must be designed in such a way that it allows to be ported another operating platforms having another mechanisms for configuration and versioning management. It will thus result in easy validation and thus more widespread application.

### 3.7 Additional Requirements Supporting Software Project

Software engineering is characterized as labor-intensive and is involved several people probably situated in different physical locations. The complex and huge technical work is often needed to be closely cooperated, and the products must be effectively shared. The engineering or design work are human-based, i.e. it involves creative and mainly non-automatic activities. Due to the complexity, software engineering activities are usually performed within context of a project consisting of several people with necessary skill and competence across divisions within an organization.

It is therefore important to manage and utilize resource and personnel efficiently. To be able to do that, it is necessary to extend the modeling concepts to embody organizational aspects, and provide
appropriate support for project management. In the following, we will treat in details issues needed to be addressed by a PSEE for better supporting software engineering projects.

### 3.7.1 Data Management

To facilitate effective sharing of data (including both products and processes), it is necessary to permanently store, and allow easy access to relevant information when performing software projects. The requirement on data persistence is of importance when working products should be available to any one at any time. It is also important to determine the degree of granularity for representing product from the viewpoint of process support. It can vary from a "atomic" level, e.g. token level of programming language, to item/module level, and to file objects. Software engineering is an experimental science whose newly developed products are normally an improvement of earlier ones. That is, data should be stored in identifiable versions for easy access. A complete system is often consisted of a fixed number of components which are versioned. Data is therefore organized in such a manner that configuration of a system can easily be extracted consisting of components with required versions. Products depend on each other in various ways - by transformations, composition, (use of) interfaces, etc. It is therefore necessary to explicitly model such dependencies properly. Some dependencies are explicit within items (cf. data granularity above), others with separate representation (module interconnection language), some are realized by activity (e.g. derivation).

### 3.7.2 Cooperation

As many people involve in working towards the same goal, it is frequent that they work on or share the same product. To effectively support for optimal work efficiency as well as no conflicting updates arising, the cooperation issue are taken seriously. Cooperation can be regarded to consist of issues related to coordination, communication and collaboration.

#### Coordination

Traditional transactions fulfill ACID\(^1\) properties, and use strict access management policies to preserve data consistency. This policy is too strict for software development. Software engineering transaction is required to relax such properties to support cooperation and work independence. Coordination concerns with data access and cooperative work problems viewed from a project viewpoint. Careful planning is therefore crucial to find out a balanced approach that allows flexible development, and still keeps consistency. That is, potential conflicting activities are detected, coordinated and scheduled in such a way that minimum number of conflicts arises. Conflicting works may e.g. be broken down and delegated to different persons to avoid conflicting updates. Coordination can therefore be characterized as preventive activities supporting cooperation.

\(^1\) (A)tomicity, (C)onsistency, (I)solation, (D)urability.
Communication

When working in a group of people, effective communication constitutes an crucial part for productivity and thus progress of software project. Such communication should provide two basic features, i.e. sending and receiving messages between work units. Communication therefore solves the cooperative problem viewed from the activity viewpoint. The message passing mechanism can support either asynchronous or synchronous transferring. The former means that the sender can send a message and continue doing other actions, while the latter requires an acknowledgment from the receiver before doing further actions. We distinguish two types of communication: all-to-all and point-to-point communication. The first communication type is required to dynamically specify the destination, while the latter statically determines the communication link between two activities (sometimes referred to as feedback). Software development normally involves a large number of predefined feedback loops, as a result of either a tool invocation or human decision. Both communication types are necessary to be supported.

Collaboration

Despite spending effort in preventing conflicts (cf. coordination above), it is impossible to predict all possible conflicts in advance, and then coordinate work correspondingly. It is thus necessary to provide flexible support dealing with unanticipated conflicts. That is, agents must collaborate when updating either same or dependent products. Collaboration therefore concerns cooperative issue viewed from product viewpoint. Any overlapping in product space must be detected, and involved human agents should negotiate to an agreed-upon cooperation protocol. Such protocols explicitly specify how a conflict is resolved by a cooperation policy. The cooperation policy can e.g. allow immediate reconciliation of changes, or restrict conflicts by enforcing serial work order. Between those two extremes, there are many degrees of flexibilities which can be negotiated. Collaboration supports must therefore facilitate negotiation and enforcement the cooperation protocol for effectively resolving conflicting changes to shared products.

3.7.3 Modeling Organizational Aspects

As mentioned above, it is important to formally describe complex organization structure as well as its related components. By doing this, better control and management can easily be attained. We can distinguish two conceptual classes of entities needed for organization modeling:

Real entity: The conceptual framework is required to be extended with new concepts necessary to reflect the organization. More specifically, new modeling notations must be able to describe organizational units like corporate, division, department, working group, ... In addition, organization's resources in form of machine, personnel or properties should also be modeled by means of the new concepts. Those entities are physically existent and therefore termed as real entities.

Imaginary entity: In the contrary, role, personnel skill, responsibility, resource allocation are volatile and their change frequency is essentially higher than real entities. Despite of such volatile nature, the imaginary entities also need to be described and covered properly by the new concepts.
Having the above taxonomy in mind, we discuss in depth the concrete organizational modeling components. We certainly need to extend the basic modeling concepts to describe following organizational infrastructure.

**Division:** Organization is often divided into groups of different sizes reflecting their functions, expertise or physical location. It can be done in hierarchical or hybrid manner. The former results in a static grouping of personnel, sometimes called line organization. The latter represents rather a dynamic organizational structure, usually referred to as project or matrix organization. Such an organization employs projects by staffing personnel resources from the static hierarchical line organization. Employing project members in such manner ensures an allocation of best suited and required skill for the project goal.

**Project:** It becomes more and more common that complex and long-durable software development is carried out in the context of project. The main reason is a one-of-a-kind property, cooperation necessity and diversity in required skill from different areas.

**Skill:** To easily identify available competence and effectively assign suitable expertise to a particular project, it is necessary to model skill explicitly. Skill is associated to a broad spectrum of entities, ranging from organization, group, to individuals. Skill is updated as maturity and capability are increased through personnel training or improved insight. The former is easily evaluated, while the latter is hard to quantify.

**Role:** As software activities are characterized as complex, it is necessary to decompose them into smaller units suitable for a group of persons or individual with appropriate competence and knowledge. A specialized skill within a given domain is often associated with a role, such as analyst, designer, tester, reviewer ....

**Access right:** We distinguish between access right to shared products and to tool/activity. Such an association between personnel or role to the access or performance authority must be explicitly represented.

**Responsibility:** A set of expectations are attached to a specific role. They can be activities which the given role is authorized and supposed to perform. To model such responsibility information is useful when determining role boundary and avoiding omission of important tasks.

### 3.7.4 Supporting Project Management in a PSEE

We distinguish project management in three distinct areas. They are project preparation; project execution and project completion. Below, we will discuss relevant requirements for each of them in details.

**Project Planning**

The planning process is a management activity usually performed by project managers, that states costs and time constraints on an up-starting or already running project, (re)organizes work breakdown and resource allocation, and enables cooperation among teams of developers that possibly work concurrently [Rom86a].
3.7. Additional Requirements Supporting Software Project

Project planning is carried out at initial phase of project and recognized to be the most critical part of development process. A development process is chosen and activities are scheduled to constitute a project plan. Forthcoming activities in the project must be scheduled over time, i.e. an original estimate on required effort is given. Information in the project plan is unique due to a vast difference between projects in complexity, platform, user environment, personnel skill, budget, .... Furthermore, the project plan is subjected to constant change as a result of unanticipated evolution occurred during project.

The initial project plan corresponds to an enactable process model in a PSEE. The process model instantiation in section 3.6.3 must take into consideration the scheduling aspect of the project plan. That is, time, cost and duration information should be integrated into the enactable process model. Such information may be automatically derived by reasoning about the process parameters such as budgetary, schedule, available resources and tools. This automatic planning probably perceive as hard to carry out, but it generates at least a plan suggestion which can be further adjusted by human. That is, appropriate support are needed to adjust the enactable process model before it is executed.

Project Execution

When the project plan is approved and its necessary resources are allocated, the project is then started. Viewing from the PSEE's viewpoint, the project plan is at this moment represented by the enacting process model variation. When carrying out the project, there are two issues which are of relevancy to be taken into consideration.

Effort tracking or monitoring: It is important for project to stay in control with respect to cost and time. We need to have mechanisms to collect actual performance measures and compare them to the initial estimates in the project plan. That is, the process model enactment (in section 3.6.4) must provide support for tracking and collecting the actual effort spent by activities in the enacting process model. The collected actual efforts are compared against the estimated ones. If the process enactment engine detects any discrepancies, it should notify the project managers for adjusting the project plan (see below). Most common project performance measures to be collected are e.g. consumed and remaining time and effort for a particular activity.

Project plan adjustment: The project plan is constantly subjected to change due to e.g. the mismatch between estimated and actual effort as described above or unanticipated change requests or pressures from the operating environment. It therefore requires that the enacting process model must be able to updated accordingly to reflect those new changes to both project plan and project constraints. Adequate support to deal with changes to the enactable process models must be fully provided by the PSEE.

Project Completion

When a project is completed, either as a fiasco or success, there exist a great deal of useful experiences which need to be packaged for future projection. What kind of experiences to be collected and packaged are dependent on the strategic objective of the organization. If the organization wants to improve its estimation capability, it should collect and compare estimated and actual effort during projects. If the fault reduction is of primary concern, it should collect and analyze fault related information such
as where faults are injected, how much effort is spent to remove them, how critical or severe they are with respect to the entire system, etc.

The modeling concepts of the PSEE should be extended to store such experiences. Moreover, adequate tool support must be provided to collect and analyze of project experiences to identify improvement initiatives or direct future action.

3.8 Requirements for Managing Software Process Evolution

In chapter 2 (State-of-the-art), we have investigated two types of software process evolution, i.e. planned and unplanned evolution. The planned process evolution is also referred to as process improvement in which systematic models and approaches (e.g. ISO 9000, CMM, QIP, BOOTSTRAP, SPICE, ...) are adopted to make the process better. An improved software process expects to develop better quality software systems within planned budget and schedule. The unplanned evolution implies changes to both software systems and software process. Evolution of software systems are caused by e.g. new customer requirements, changing operating environment and so on. This dynamic property of software systems has been studied by Lehman and Belady for a long time. Unplanned evolution of software process is an inherent consequence of the evolutionary world software process is operating on. Such unpredictable evolution of software processes is inevitable and occurring more frequently than the planned one. The direct impact of such unanticipated evolution is that the project schedule, its estimated cost, its resource allocation, its duration and etc. are all affected.

3.8.1 Evolution Classes

In the following sections, we will discuss how evolution of software process can be effectively addressed in the context of a PSEE. First, we would like in this section to clarify terminologies used as software evolution is concerned.

Figure 3.2 illustrates three classes of software evolution.

**Software system evolution** includes changes and growths to an information system which is embedded in the application domain based on change requests or new insights. The information system is built to support and guide enterprise in performing their activities efficiently. However, the requirement on the information system is revised over time as new improved insights are gained. The information system which wants to survive, must therefore evolved to adapt to and incorporate new needs. The thorough study of software system evolution or dynamics has been conducted by Lehman and Belady for a long time.

**Process evolution** denotes changes to real-world software process (including production process and meta-process). The information system is developed and maintained by the production process. The meta-process manages and evolve production process, itself and their corresponding process models. Process evolution is the most critical and difficult task of all. As mentioned above, there are two types of process evolution: planned and unplanned. Planned process evolution (e.g. process improvement) is carried out by first describing new changes to the generic process models. New changes are then migrated to newly created enactable and enacting process models which guide and support the real-world process performance. We thus assume that development
3.8. Requirements for Managing Software Process Evolution

Figure 3.2: Software Evolution

An organization consistently uses PSEE to manage and enforce its processes. With that assumption, we simplify the transition of new model changes to the real-world. However, such process evolution is still complicated because it involves human being, work habits, team morale, social relationships, etc. Unplanned process evolution is often caused by unexpected circumstances which requires the process to be adjusted. Such adaptation of processes must be reflected in the corresponding models to sustain the consistency of the PSEE.

**Process model evolution** refers to changes to the process models in a PSEE. As described above, the planned process evolution implies changes to the generic process models, and new changes then propagate to enactable and enacting ones. We choose to refer to such model change as *top-down* process model evolution. On the other hand, unplanned process evolution often occurs within a project, and requires the enacting process model to be adjusted. A project is a specific instantiation of a process model, i.e. the enacting process model variation. Sometimes such adjustments to enacting process model does not necessarily lead to modification to the corresponding generic one, and are therefore carried out by explicit demand. It often involves change directly to the project schedule, resource, personnel, budget, constraints, .... We refer to such model change as *bottom-up* process model evolution.

We are in this thesis focusing on the process evolution and process model evolution issues. The requirement to PSEE’s PML in section 3.6.2 has stated that process models (including three model variants) should be evolved. The adjustment to enacting process model (or bottom-up process model evolution) corresponds to the requirement for project plan adjustment in section 3.7.4.
Chapter 3. Objectives and Requirements

There exist a large number of software process improvement methods/models (see section 2.3), and supports provided by several PSEEs (see section 2.2.3) to address and deal with planned process evolution and top-down process model evolution respectively. However, the problem of unplanned process evolution and bottom-up process model evolution is still poorly supported by any PSEEs. In the following sections, we will discuss requirements to manage those two evolution aspects in details.

### 3.8.2 Bottom-up Process Model Evolution in PSEE

A PSEE should provide flexible support for evolving the enactable and enacting process models according to changes which are detected before and during project execution. It is common that changes to project parameters (e.g. estimates, budget, schedule, resource, tool, etc.) can occur both before and during the project actually starts. Existing project management tools offer supports for performing changes to the original project plan, and for revising project parameters during execution (see section 2.6.3). However, they are mostly performed by human which requires to have sufficient knowledge and skill. The discussion in section 2.6.4 has advocated for an integration of project management facilities in the PSEE to better exploit available process knowledge and experiences.

### 3.8.3 Unplanned Process Evolution in PSEE

To effectively deal with unplanned process evolution, we need to obtain better understanding about its nature and impact. We need to get a more structured insight into the problem before we can analyze and then suppose preventive initiatives. Our objective is to be able to better anticipate and predict such evolution before and during project. That is, we then get more control over the project by ensuring the minimum of surprises along the way. As DeMarco in [DeM82] states: "The best-controlled project is not necessarily the one that does the best or most work, but the one that best lives up to its predication. When there are deviations from what the project originally proposed to deliver, those deviations are minor and they are signaled early."

In the following, we outline some requirements necessary to be addressed to better manage unplanned process evolution.

**Better understanding** of the properties of the unplanned process evolution. Such understanding can be achieved by classifying evolution occurrences. Data collection form completed and ongoing projects is a suitable approach for better understanding evolution phenomena. Further, the evolution data which are collected, must be mapped into a structured framework so that patterns can be recognized. The evolution impacts is also useful to be recorded.

**Thorough analysis** of the collected evolution data to recognize their commonalities or differences. When the evolution data are collected from a acceptable number of project, they need to be analyzed carefully. By doing so, the evolution patterns are recognized and prioritized upon the severity of their impacts. In addition, relevant empirical relations between evolution and other project/product characteristics must be identified. To do so, the evolution data must be formally recorded to facilitate reasoning. Recognized empirical relations help to identify which evolution patterns are most likely to be occurred within a given project and product profile.

**Reuse** of recorded evolution data and its associated analytical information to effectively anticipate and predict impact of evolution in similar projects. It is necessary to learn and reuse past expe-
3.9. Chapter Summary

In this chapter we have surveyed the general and overall objectives concerning research on software processes. They concern with e.g. process assessment/improvement, simulation, learning and support (PSEE). Our underlying research objectives are then identified as:

**O1** Integrating SCM and project management into PSEE.

**O2** Managing process evolution within PSEE context.

**O3** Validating PSEE.

Some relevant research issues are then identified relating to production process, meta-process and process support. On the basis of those recognized research issues, we state a set of requirements for our work in the remainder of the thesis. Stated requirements covers three main areas of concerns considerably relevant for a PSEE. The first area includes basic requirements which address issues such as modeling concepts; instantiation and enactment of process models. The second area describes additional requirements concerning to software development project. They are dealing with issues like data management, cooperation (including coordination, communication, collaboration), organizational modeling and project management. The last area of concern addresses required support for managing software evolution during software project. The primary work objectives concerning management of process evolution are:

- Increase the comprehension and awareness of continuous process evolution and its consequences by systematically recording, learning, and assessing unplanned process evolution during projects.

- Improve management practices, e.g. to better plan, estimate, schedule, follow-up, and evolve software project (i.e. bottom-up process model evolution), by reusing evolution history of other similar completed projects.
Chapter 4

EPOS Baseline

4.1 Introduction

In this chapter, we describe in details architecture and features of EPOS as a PSEE prototype. EPOS stands for Expert System for Program and (Og in Norwegian) System development. This project had been supported by the Royal Norwegian Council for Scientific and Industrial Research (NTNF) through grant ED0224.18457. The EPOS project was from 1986-1990 a joint R&D project between our Software Engineering group and ELAB-UNIT in Trondheim, and Veritas Research and Sysdeco on Oslo. The initial and primary project goal was to implement a software engineering database exploiting participants’ expertise on Object-oriented (OO) technology, OO database, and configuration management (CM). It resulted in a prototype of EPOSDB. After that, the EPOS project has been partly funded by Norwegian Research Council (NFR) through the REBOOT project. The process management (EPOS-PM) began first then to be focused in the project. The entire project relies on resources from funded PhD students, and under-graduated students through various projects.

The chapter is organized as follow. First, we present an overview of the EPOS system followed by the overall architecture in section 4.3. We then elaborate in details the two layers of the EPOS system, namely EPOSDB and EPOS-PM in sections 4.4 and 4.5 respectively. A summary of the contribution is outlined in section 4.6. Finally, a temporary evaluation of the EPOS system against the stated requirements in chapter 3 is presented in section 4.7.

4.2 Overview

To guarantee a most simple introduction of our complex system, we describe below an overview of EPOS avoiding the use of too many technical terminologies. However, the following description will gradually become more technical as the reader proceeds through the chapter.

EPOS [MCL+95] [CHL+94a] is a kernel, software engineering environment supporting both software configuration (EPOSDB) and process management (EPOS-PM). Figure 4.1 depicts an overview of EPOS as a PSEE prototype which models real-world software processes and then assists their performance.

The real-world software processes include descriptions of both production and meta-process activities. The former describes software engineering and managerial activities, while the latter describes how the entire software process is evolved. People involved are allocated according to an organiza-
tion structure, and assigned to participate in one or several projects. A project is a collection of related and inter-dependent activities which applies a set of production tools to develop products (i.e. software artifacts). Most of the products are represented by tangible files in workspaces (i.e. directories in a file system). Those files can be shared between different people and projects. To properly support and guide the performance of real-world software processes, relevant parts (or preferably all) of those above are represented by internal process models in EPOSDB. A project is represented by a decomposable networks of activities (i.e. task networks). The hierarchy of such task networks facilitates both work breakdown, delegation and sharing common products and tools. The internal process models and task networks are instrumented and executed by a set of EPOS-PM's process tools. By executing the task network, the real-world software processes are guided and performed efficiently.

We apply a task network formalism in modeling real-world software processes in EPOS. That is, an engineering or managerial activity is represented by a task which is considered as a basic component of the task network. Each task has a defined number of inputs and outputs which represent tangible products or deliverables. Figure 4.2 is a snapshot of the task network depicted in figure 4.1 above.

The primary goal of the task is to provide outputs in expected states by consuming available inputs. The inputs consumed by the associated task must satisfy a set of constraints or conditions (entry criteria). In addition, the outputs must reach certain states (i.e. exit criteria) before the task is declared to be completed. To be able to make the outputs obtain such states, the given task contains a sequence of actions which possibly invokes external production tools (e.g. editor, compiler, ...), or initiates an interaction with human agents, or combination of them. Occasionally, the task represents a complex activity. Such a high-level task is required to be divided or decomposed into several simple subtasks which can recursively be decomposed further. An example of a high-level task can e.g. represent a wa-
terfall life cycle model, or simply an implementation phase of the former. The entire task network (e.g. in figure 4.2) is an instantiation of a defined internal process model. The process model can be instantiated into different task network variants (i.e. with different layout) depending on the issued goal to the outputs of the root task. EPOS automatically performs such instantiation by reasoning about such issued goals. The generated task network is then interpreted and executed in both sequential and parallel manner. The defined entry conditions associated to task inputs are evaluated, and then determined whether the given task is ready to be executed or not. By executing tasks in the task network, human agents are guided and assisted in performing the designated activities efficiently. When carrying out activities in the real-world, human agents gradually get matured and possibly identify undesirable or deficient properties in their processes. EPOS thus provides adequate support to evolve existing process models reflecting the improved insights.

In the following, we conclude the overview with a summary of EPOS described in technical notations. As mentioned above, EPOS supports integrated both software configuration and process management through a two-layer architectures:

- At the bottom level, a client-server architectural EPOS database (EPOSDB) offers a traditional ER data model extended with object-oriented properties such as, subtyping and meta-types. In addition, the EPOSDB provides uniform versioning, and nested and cooperating transactions. EPOSDB is built to store models of versioned software products, as well as their software processes.

- At the upper level, EPOS-PM supports a reflexive, object-oriented software process modeling language (PML), called SPELL to describe persistent and versioned products, activities, tools, resource, project, and their meta-process (i.e. policies or rules describe how to manage processes). SPELL extends the underlying data model with behavioral type-level properties, procedures, triggers and tasking. To support process management, we have designed and implemented a set of process tools facilitating modeling, (re)planning, enacting and evolving the process models. Those features manipulating process models will be treated in details in following sections. We have also elaborated low-level transaction planning mechanisms for better coordination of cooperative work, and some primary support for simple project management practices.
4.3 EPOS Architecture

In this section, an overall architecture of EPOS is summarized. The underlying design and implementation decisions of EPOSDB and EPOS-PM are presented in sections 4.4.1 and 4.5.1 respectively. They clearly explain the rationale as well as the justification on which the architecture is elaborated. Within the description below, important issues which are elaborated further, are explicitly referred by section numbers in the chapter. Figure 4.3 illustrates the two layer architecture mentioned above. The EPOS-PM layer encompasses from layer 2 to 5.

1. **EPOSDB: a OO client-server DBMS** is implemented on UNIX operating platform and C-ISAM. The EPOSDB kernel consists of four layers. i) low-level records; ii) uniform change-oriented versioning - COV (section 4.4.2); iii) structurally object-oriented data and transaction model. Long, nested and cooperating transactions define a single version context for data access. (section 4.4.5); and iv) Object access interface. Remote Procedure Call (RPC) is adopted to support communication between server and clients. EPOSDB has a simple Data Description Language (DDL) and a free-standing Data Manipulation Language (DML), both in Prolog. An EPOSDB transaction is connected to a file-based workspace, through check-in/out or DB/WS mapping (section 4.4.3). The EPOSDB provides a set of kernel types (section 4.4.4) for instrumenting types in following layers.

2. **A reflective and fully object-oriented PML**, called SPELL, that unifies and extends the underlying DDL/DML. Such extension is represented in a set of new types. The language constructs, syntax and properties are described in section 4.5.3 together with its interpreter.

3. **A Tasking Framework** for definition, instantiation, and concurrent execution of task networks, with associated basic process tools such as Planner (section 4.5.5.2) and Process Engine (section 4.5.5.1). They are interacting closely to construct and enact the task network for guiding and assisting process performance. A set of types are defined to support the tasking framework.

4. **Advanced Process Tools** They include a set of embedded process tools providing advanced support for process management. A Schema Manager (see section 4.5.6.1) offers support for defining and evolving process models. A Task Network Editor (section 4.5.6.2) is a graphical canvas for drawing required task networks by which necessary process models are then generated. **CHAT** which provides flexible supports for handling conflicting updates, is presented in section 4.5.6.3. A Transaction Planning Assistant (section 4.5.6.4) offers intelligent support for coordinating EPOS transactions effectively preventing future update conflicts. A Broadcast Message Server (BMS) supports communication between process tools and external tools (in section 4.5.6.5). Simple project management support is provided by a **Project Manager Tool** (section 4.5.6.6).

5. **Application Process Models**. They include Process Schemas, (e.g. new types) to augment the kernel types which are defined in the four layers above to model real-world process applications. Examples of such augmenting types can be found in our modeling experiences in chapter 5. Some predefined types (see section 4.5.4) which are stored in the EPOSDB, are instrumented and executed by EPOS-PM's process tools described above.

All those five layers above constitute an PSEE. A Graphical User Interface (GUI) is also provided to interact between the EPOS user and both basic and advanced process tools (i.e. layer 3 and 4). In the
following sections, we describe in details all layers and their components. To easily navigate within the chapter, table 4.1 below indicates where important topics can be found.

4.4 EPOSDB

In this section, we present the original versioning approach called COV in 4.4.2 followed by the version/configuration selection approach adopted in EPOSDB in 4.4.3. The EPOSDB kernel data model and transaction model are then presented in 4.4.4 and 4.4.5 respectively. The issue of propagation semantic and cooperation are further elaborated in section 4.4.5. The communication between EPOSDB transactions is presented in section 4.4.7.

4.4.1 Underlying Design Decisions for EPOSDB

Some of the basic decisions underlying the design of EPOSDB are given here. The intent is to provide reader the rationale of our technology and solution choices. The EPOSDB prototype has undergone three major re-designs and as many versions has been released. The total volume of the EPOSDB is about 25K lines of C-code.

Availability of knowledge The local knowledge about object-orientation technology and its database is strongly represented at our group, especially with the active participating in the REBOOT
<table>
<thead>
<tr>
<th>Layer</th>
<th>Topic</th>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>EPOSDB</td>
<td>4.4</td>
</tr>
<tr>
<td>1</td>
<td>Versioning (COV)</td>
<td>4.4.2</td>
</tr>
<tr>
<td>1</td>
<td>EPOSDB Data Model</td>
<td>4.4.4</td>
</tr>
<tr>
<td>1</td>
<td>EPOSDB Transaction</td>
<td>4.4.5</td>
</tr>
<tr>
<td>1</td>
<td>Transaction Communication</td>
<td>4.4.7</td>
</tr>
<tr>
<td>2</td>
<td>SPELL</td>
<td>4.5.3</td>
</tr>
<tr>
<td>2</td>
<td>SPELL Interpreter</td>
<td>4.5.3.3</td>
</tr>
<tr>
<td>3</td>
<td>Tasking Framework</td>
<td>4.5.5</td>
</tr>
<tr>
<td>3</td>
<td>Process Engine</td>
<td>4.5.5.1</td>
</tr>
<tr>
<td>3</td>
<td>Planner</td>
<td>4.5.5.2</td>
</tr>
<tr>
<td>4</td>
<td>Advanced Process Tools</td>
<td>4.5.6</td>
</tr>
<tr>
<td>4</td>
<td>Schema Manager</td>
<td>4.5.6.1</td>
</tr>
<tr>
<td>4</td>
<td>Task Network Editor</td>
<td>4.5.6.2</td>
</tr>
<tr>
<td>4</td>
<td>CHAT</td>
<td>4.5.6.3</td>
</tr>
<tr>
<td>4</td>
<td>Transaction Planning Assistant</td>
<td>4.5.6.4</td>
</tr>
<tr>
<td>4</td>
<td>Broadcast Message Server</td>
<td>4.5.6.5</td>
</tr>
<tr>
<td>4</td>
<td>Project Manager</td>
<td>4.5.6.6</td>
</tr>
<tr>
<td>5</td>
<td>Process Models</td>
<td>4.5.4</td>
</tr>
<tr>
<td>5</td>
<td>Activity Model</td>
<td>4.5.4.1</td>
</tr>
<tr>
<td>5</td>
<td>Tool Model</td>
<td>4.5.4.2</td>
</tr>
<tr>
<td>5</td>
<td>Product Model</td>
<td>4.5.4.3</td>
</tr>
<tr>
<td>5</td>
<td>Resource Model</td>
<td>4.5.4.4</td>
</tr>
<tr>
<td>5</td>
<td>Project Model</td>
<td>4.5.4.5</td>
</tr>
<tr>
<td>5</td>
<td>Meta-process Model</td>
<td>4.5.4.6</td>
</tr>
</tbody>
</table>

Table 4.1: Section Overview

project [Kar95]. Such availability of knowledge has led us to select an OO design and implementation of the EPOSDB.

**Open-endedness** The initial objective of EPOS project is to develop an OO, general-purposed software engineering database management system (DBMS). That is, it should be instrumentable, kernel rather than a full fledged DBMS. That is, it should be sufficiently flexible and open-ended to facilitate integration of new tools. The design of EPOSDB should also facilitate platform portability and interoperability. We have determined to adopt a combination of OO and Entity-Relationship as the data definition language (called EPOS-OOER). In that way, we can benefit from both approaches.

**COV - Original versioning approach** The research on versioning and configuration management in our group has resulted in an original, purpose-built versioning approach, called COV (see section 4.4.2 below). The COV was first introduced in [Hol88] for simply dealing with text file (source code). The major work to transfer the COV idea to a database context was carried out in [Lie90]. The full design and implementation of COV in EPOSDB has been conducted by a PhD thesis of Munch [Mun93].
Limited resources  Being an university research project, EPOS has a limited funding for both PhD students and other expenses. We need to apply either share-ware or available software at the department or commercial off-the-shelf-software. As a consequence, we have determined to use C-ISAM at lower level database and C-language in UNIX platform for implementing the user interface.

Client-server architecture  To support distributed work environment in software engineering and also ease the maintenance effort, we have applied the client-server architecture for our DBMS. That is, a EPOSDB server maintains a common database, communicates with a number of EPOSDB clients and offers DBMS services to application processes through the EPOSDB interface. EPOSDB client instances are loaded as part of application process, each may run on any UNIX workstations with an RPC connection to the server.

Long, nested transaction model  The recognition of potential shortcoming of traditional short, atomic transaction in software engineering context, we have designed a long and nested transaction model. The long and nested transaction model should better support software engineering transactions which can last for one second or several months, and involve several persons with needs for both isolation and cooperation.

Flexible cooperation support  We find our initial transaction model with fixed propagation semantics too strict and rigid to support cooperation in software engineering. The transaction model is therefore extended to include more flexible, user-defined cooperation protocols and a set of supporting tools. An approach is then introduced to avoid updating conflicts even before the transactions start. This conflict-preventive approach consists of a formal language to describe intended operations and group them in well-defined work units. Thereafter, the approach will then organize and sequence the transactions in such a manner that the extent of updating conflict are essentially reduced.

Modularity  The strict layering of EPOSDB kernel as depicted 4.3 facilitate the independence between layers. This independence is not visible for users of applications using the EPOSDB, but it means that we can easily replace the underlying storage with any other DBMS, and can also replace the transaction management and/or the top level data model with something else. Similarly, we can replace the COV internals without affecting other parts of the database.

4.4.2  COV

The idea of Change-Oriented Versioning (COV) is originated by Holager [Hol88]. Lie [Lie90] has generalized and further realized the original idea to application for database. Munch [Mun93], [LMGC93] has revised the COV paradigm for software engineering database.

In most existing configuration systems, like SCCS [Roc75] and RCS [Tic85], the most primary concept is version. Changes are secondary. We refer to those system as Version-Oriented Versioning (VOV). This view defines a configuration by firstly selecting a set of versioned components. The selection is performed by e.g., enumerating or more sophisticated manners based on attributes, constraints, etc. Secondly, the selected versioned components are bound into a specific system version.

In COV, the primary concept on the contrary is change, especially functional change associated with external characteristics of the product. Each functional change is identified with an user-defined option,
being a boolean variable whose value domain is ternary: True, False and Unset. Options are
global: an option may be used by different users, and an option may involve changes to more than one
component. However, it is not necessary that all components are affected by a single option. From the
COV perspective, a configuration is a set of functional changes, applied to the entire database. The
COV is in principle orthogonal to data model and instances. A version-choice (VC) is defined
to be a list of option settings which identifies the selected version and thus regulate the reading. An
ambition (A) is a list of option settings in which Unset values are allowed. The ambition which
is usually attached to a transaction, identifies a set of versions in which current changes will be visi-
ble. The ambition therefore defines the intended scope of the changes: if an option is left unspecified
(Unset), the changes will be visible independent of which value this option given in later version-
choices. We can say that the version-choice regulates the “reading” from the DB, and the ambition the
“writing” back to the DB. The validity is a boolean expression over options determining which
choices produce versions that are valid or completed according to some criterion.

An example will simplify the use of the COV-concepts mentioned above. A software product is deliv-
ered for IBM-PC, Sun and VAX. The Sun version may be used both on dumb terminals and under the
SunView window system. The options PC, Vax and Sun describe on which processor the product
is going to run. The option SunView indicates which window system is in use. If COV user wants to
build a Sun version with window support, a choice setting the options Sun and SunView to TRUE
is given. Now, a choice setting SunView and Vax should not be valid since Vax does not support
SunView. Furthermore, if only option SunView is given, it should be sufficient because it is only Sun
that is using SunView. The following expression will be the validity for the current situation:

MutuallyExclusive(PC, Vax, Sun) \ (SunView \rightarrow Sun)

By having this validity, the value for all other unspecified options are determined implicitly, when only
one of PC, Vax or SunView options is given.

The basic versionable unit or fragment in EPOSDB is a text line or a relational tuple. Each fragment
has an attached visibility (an expression over existing options), being maintained in the context
of EPOSDB long transactions (see 4.4.5). Visibility which is a boolean expression over options is
evaluated to TRUE or FALSE for a given choice. The result determines whether the attached fragment
will be visible or not in a particular version. That is, a version of an object consists of all fragments
with visibilities evaluated to True.

4.4.3 Configuration Description and Selection in EPOSDB

A configuration is described by a database query or intentional config-description, CD = [VD, PD].
The version-description (VD) and product description (PD) are described as follows:

Version-description: (VD) consists of a partially bound version binding. It can be evaluated to a com-
plete choice by stored version-rules, being logical expressions: constraints (static) and prefer-
ences/defaults (dynamic). Example:

Version-description VD = (Machine_Sun, Status = Tested)

The version selection will be checked against and expanded according to the versioning rules.
**Product Description:** is a tuple of [Root objects, ER types] to evaluate a Product Structure through a product-view closure on the product graph. This closure is defined by a hierarchical product model. Example:

Product description $PD = \text{ROOTS, RELATION}$,
$\text{ROOTS} = (\text{MainpropA})$
$\text{RELATION} = (\text{Contains, Depends})$

A bound *configuration*, a Config, can then be determined by a config-description (CD) and a database (DB).

$$\text{Config} = \text{Evaluate-config(CD, DB)}$$

The Evaluate-config then defines in four following mapping steps, see also Figure 4.4:

**C1. Bind-version** (VC, A) := Bind-version(VD, Version-rules)

That is, the more high-level and partially bound but incomplete VD is evaluated to a low-level and completely bound version-choice (VC), and to a possibly less bound ambition (A). The VC is a point in the multi-version space indicated by A. The mapping typically incorporates default “revision options” in the version-choice, and may change as the version-rules evolve [GKY91].

**C2. Version-select** Sub-DB := Version-select(VC, DB)

This mapping facilitates “COV-propagation” (raw data merging) *between* versions, (or configurations), if the changes have wide-enough visibilities. This is the main feature of COV, since it eliminates the need for separate, parallel variants. The Sub-DB thus contains single version of the entire database DB.

**C3. Product-select** Config = Product-select(PD, Sub-DB)

This mapping selects a closed Product Structure, i.e. a configuration, by constraining the bound sub-DB. The mapping may change, due to structural changes in the product (really a C2 change).

**C4. DB/WS map** WS := Check-out(Config, DB-WS map)

This fourth step will generate a workspace (WS) in file-based storage which allows easy access by conventional tools, e.g., text editor, compiler, etc.

As mentioned previously and demonstrated above, the version-selection (C2) is done *before* product-selection (C3), being the inverse of common practice in most CM systems.

### 4.4.4 Data Model

The EPOSDB provides an extended ER data model (see figure 4.5) with single subtyping. Named *attributes* can be defined, with the usual simple domain supported. Attributes are inherited by subtypes. A relationship type is an association between two entity types. A relationship instance is a particular link between two entities, and has no identity of its own. A relationship subtype may add attributes, and also restrict one or both roles, or restrict cardinalities.

An extension from the standard ER-model provided by the EPOSDB is that schema information is directly accessible as (read only) proper entities and relationships in the database. Each entity or relationship type is therefore represented by a type descriptor *TypeDesc* entity (or meta-type), and each
Figure 4.4: EPOSDB mappings.

Figure 4.5: ER data model of EPOSDB predefined types
attribute by an attribute descriptor `AttributeDesc`. These, and relationships between them, give user the possibility to query for schema information. It is also possible to subtype the type descriptor and e.g., add new attributes. Thus, we can add user-defined type information without affecting the DBMS semantics. The `TypeDesc` has two subtypes: `EntityDesc` and `RelationDesc` being meta-types of all user-defined entity or relation types respectively. The `AttributeDesc` has a set of subtypes necessary for representing various attribute types.

There is a predefined type `Longfield`. Instances of this `longfield` type (or subtypes) have additional operations for checkin/checkout of the file contents. A predefined type `Message` is used in communication facility between EPOSDB transactions (in section 4.4.7). The `Message` type can be subtyped, adding more necessary attributes.

### 4.4.5 EPOSDB Transaction

EPOSDB supports `long`, `nested` and `persistent` transactions, with associated sub-databases.

- Transactions are `long` due to the fact that they are used to implement an engineering change performed by a human, and typically stay alive for weeks, rather than seconds. A EPOSDB client may connect to and disconnect from a transaction. A client is at any time connected to at most one transaction, and all database operations issued by the client will be done within the context of its current transaction. Only one client may be connected to a given transaction at any time. A transaction is associated with its own versioning context: ambition + choice (see section 4.4.2). The choice must be within the ambition, and can be changed during the lifetime of the transaction.

- Transactions are `nested` and hierarchically structured where each transaction, except the root transaction, has a single parent, and may have none or many children. A sub-transaction’s ambition will “inherit” all the option bindings of the parent’s ambition, and may further restrict it by binding more options, but it may not remove or override any of the inherited option bindings. The sub-database of a sub-transaction is initiated with a subset (or possibly all) of the sub-database of the parent transaction.

- Transactions are `persistent` in the sense that they have an existence and the data survive independent of the process that created them. This persistence property facilitates history tracking of completed transactions.

#### 4.4.5.1 Basic Change Propagation Semantic at Commit

A EPOSDB transaction has an ambition and a version choice, as mentioned above. The version choice must be within the ambition, and can be changed at any time (see figure 4.6). The ambition can not be changed once the transaction is started. The ambition specifies the scope of changes in the transaction. The ambition has possibly `Unset` option bindings, and may thus identify a potentially large set of versions or configurations, where changes from the current transaction will be visible after commit. EPOSDB transactions do not operate in total isolation until commit. Rather, there is change propagation between transactions. Such change propagations are automatically initiated at transaction commit, but explicitly invoked by user during transaction (pre-commit).
There are basically two ways of propagation: from parent to child, and from child to parent. All other propagation paths are combinations of more than one step of these. Note that updates to locally created objects will not be taken as normal updates, but will simply replace the first insertion with another one. An update to an already (locally) updated (sub)object will replace the first update. Likewise, a deletion of a locally inserted object or relationship instance will be done by simply removing it. Only when we change something that is not totally “ours”, will we have to think about the effects.

A “golden rule” behind these semantics is that whatever we see in a sub-transaction shall be exactly the same as what the parent transaction would see, if we first did a commit and then looked at the data from the parent transaction, with the same version choice (Or for that matter, if we start a new sub-transaction and set the same choice.)

**Parent-to-Child** The basic rule is that everything done by the parent is immediately visible by all child transactions. The rule applies recursively down the transaction tree.

**Child-to-Parent** Upward propagation, from child to immediate parent, is done at commit or pre-commit. Of course, nothing is propagated at abort. Any change that is propagated to the parent will be written as if it were done by the parent itself, so that all rules for downward (i.e. parent-to-child above) propagation to siblings of the committed transaction apply. The only difference is that the effect is restricted by the ambition of the committed transaction. It should be stressed that committed changes will overwrite conflicting changes done by the parent or by previously committed sibling transactions, conforming to the “last wins” or “copy-on-write” principle, though (as said) restricted by the ambition of the committed transaction. The parent may in turn overwrite the committed changes.

Such a propagation semantics above is too strict for software engineering activities. We need to negotiate and eventually resolve updating conflicts before commit. That way, we prevent from loosing important changes. That is the reason why we need more sophisticated cooperation protocol managing change propagation between transactions (see in section 4.4.6 below)
4.4.6 Cooperation Support in EPOS

Cooperation and coordination, by information sharing and exchange, are essential in the use of database support in engineering application. When several activities use the same parts of the product, we need to coordinate access to the database to avoid conflicts and inconsistencies. There are two overlapping cases where cooperative support is necessary (described in COV concepts):

- **Ambition-overlap**: mutual version visibility.
  Two ambitions are *overlapping*, if no option is *False* in one ambition and *True* in another (but *Unset* is OK). Otherwise they are *disjoint*. No ambition-overlap corresponds to classical variants.

- **Product-overlap**: shared subsystems.

The degree of overlap can be used to assess the potential impact (side-effects) of the intended changes of a new transaction. A consequence can be that the ongoing transaction must be constrained, i.e. its ambition narrowed. Alternatively, the new transaction can be constrained, delayed (i.e. serialized!), or delegated to somebody with proper access rights. Such solutions seem too restrictive for software engineering activities.

4.4.6.1 Cooperation Protocol

We have therefore in [NC93] defined propagation protocol which offers flexible cooperation between transactions in EPOS. Such agreed-upon propagation protocol thus reconciles updating conflicts in an user-defined manner before commit.

**Propagation protocol** is established when two transactions are detected as overlapping. The propagation protocol for transaction \(T_i\) and \(T_j\) contains following information:

- **Granularity** denotes *what* kind changes done by either of those two transaction will be propagated to the other. It can e.g. be selected types of products or the entire product structure (i.e. configuration).

- **Timing** specifies *when* the transaction wants to receive and implicitly when the other transaction will send change notification. It consists of three values: i) *busy* - changes are propagated immediately; ii) *semi-busy* - changes are propagated at manual demand (e.g. every hour, at the end of the day, weekly, ...); and iii) *lazy* - changes are only propagated after own, or other's commit.

- **How to accept** describes how transaction reacts upon change notification sent by its partner. The user reaction is distinguished into two manners: i) *accept* - changes will be either automatically or manually incorporated into the private workspace (see how-to-reconcile below); ii) *reject* - can be refined in: delay (ok, but not-yet-ready), veto (return protest message possibly with proposed changes), reduce (mutual version visibility).

- **How to reconcile** specifies how new changes will be incorporated if the user accepts to reconcile changes done by its partner. It can be carried out by two ways: i) *auto-copy* - shared files or indirect file link; ii) *manual-copy* - e.g. manual merging of two files.
An add-on cooperation process model defined in EPOS-PM, provides support for establishing propagation protocol, and using it to manage updating conflicts. A more detailed explanation of this cooperation model is found in our modeling example ISPW7 in section 5.4. Thereafter, we determine to embed cooperation support as a feature into EPOSDB, rather than an external process model. It yields that a more comprehensive cooperative transaction model is defined (see section 4.4.6.2 below).

### 4.4.6.2 Cooperative Transaction Model in EPOS

We want the EPOSDB to be able to support a wide range of work modes which again will require different types of cooperation support: At one extreme, users may use conservative two-phase locking to enforce a serial execution of workspaces. At the other extreme, it is possible to allow full concurrency during transaction execution, and then, in case of conflicts either undo the changes or force the users to merge conflicting, concurrent changes.

In order to to that, we want the EPOSDB to be able to provide users with more information about other users’ current and planned work, spanning from identification of which objects are being changed to user-oriented descriptions of work intentions. To allow for more active cooperation, the system should be able to give its users notifications about events caused by other users. In addition, the system must be extended with operations for more flexible sharing and exchange of objects between users working in different workspaces. The existing EPOS transaction model (in section 4.4.5) is extended with following support for cooperative work [Lar94]:

**Declaration of work intention** Planning information (e.g. resources, calendar, dates, and expected work sets) are specified in a formalism comprehensible to both users and the system. These specifications are primarily meant for a priori coordination of work. The term *work-unit* (WU) is introduced to cover the notion of a “meaningful”, logical unit of change/revision made to software objects in the EPOSDB. WUDL (Work Unit Description Language) is a textual language for defining properties of and pre-declaring work intentions for a workspace or transaction. A WU specification is stored in EPOSDB, and accessible by other users so that it will be possible to coordinate and schedule larger changes jobs to minimize the potential for access conflicts. We have also designed a planning tool [CLH95] (see also section 4.5.6.4) which can analyze a set of WU specifications and suggest an execution history which will minimize the set of dependencies between concurrently executing transactions.

**Transaction information** Such information is stored in the EPOSDB and maintained when users are performing operations related to transactions. Some examples of the information are: a general description of the work intention, the transaction structure (hierarchy), and the set of accessed objects. This information is accessible through a set of tools/commands, and can be used to detect and/or avoid access conflicts at any time.

**Event and notification** Whenever the user invokes a command that will update the database contents, a *notification message* is sent to all users that may be affected by the event. The notification messages are displayed and managed through a separate user interface, with features for filtering and user subscription to message classes (cf. CHAT in section 4.5.6.3). The critical events which need to be notified are e.g. possible or actual write/write conflict. Such events are associated to common changing operations in the EPOSDB such as: update, lock, check-in, check-out, create, delete software components.
Flexible object propagation The transaction model has been extended with operations allowing flexible exchange of objects between transactions. These operations are available to the EPOSDB users through additional commands.

4.4.7 Communication between EPOSDB Transactions

This communication is offered by EPOSDB to send messages between EPOSDB transactions. Such communication is performed by sending an instance of type Message or its subtypes from one transaction to another. The communication is asynchronous; i.e., there is no waiting involved at either end. The message is an instance of the predefined type message, to which additional information can be added by creating subtypes with the necessary attributes. The database provides function to:

- Create and send a message to a specified transaction. The message will not be copied locally in the sender transaction.
- Test for the existence of messages to the current transaction. This is a non-destructive test; no message is actually read.
- Receive the first message in the incoming “mailbox” of the current transaction. The message will be removed from the mailbox and become a readable entity.

The predefined message type includes three pieces of information provided by the database when the message is created. They are the sender ID, receiver ID, and a time stamp. User can define new subtypes which can add necessary attributes.

4.5 EPOS-PM

In this section, we present a detailed architecture as well as the functionality of the EPOS-PM layer. At first, we present our Process Modeling Language SPELL, including its language constructs, properties and interpreter in section 4.5.3. We then describe different process models representing activity, production tool, product, resource, project and meta-process in section 4.5.4. In section 4.5.5 basic process tools for tasking framework such as Planner and Process Engine are presented followed by a description of other advanced process tools in section 4.5.6.

4.5.1 Underlying Design Decisions for EPOS-PM

Some of the basic decisions underlying our design of EPOS-PM are given here to illustrate the rationale of work. The original of implementing EPOS-PM is to build an add-on application on top of the EPOSDB. The intent of building process support on top of EPOSDB is twofold: i) validating the configuration management facilities offered by EPOSDB; ii) aiming to develop a fully integrated environment providing support for both configuration management and process management. The following decisions describe the sound inference process on which the design and implementation of EPOS-PM essentially rely. The first four points relate to underlying process modeling formalism and process model description. The next two points involve process instantiation or creation. The next two points concern
the execution or enactment of process model while the last point deals with architectural solution for model evolution.

**Rule-based paradigm** We will mainly use rules to represent process knowledge in EPOS-PM. The decision is made based on the fact that such rule-based representation provides flexibility, ease of modification and interpreting feature. We argue that a set of rules can be easily maintained, modified and dynamically interpreted. However, we also realize potential problems with rule sets such as: structuring and modularizing a large number of rules/facts; and anticipating the (side-)effect of rule set modification.

**Tasking framework** or net-oriented paradigm is consisted of both vertical work decomposition and horizontal product/activity chaining. Here, we argue why they are suitable for software engineering. It is well recognized that software engineering deals with a complex and intangible problem. The traditional, standard technique *divide and conquer* is often used to reduce the complexity and increase the clarity of the problem. On the other hand, most life cycle models involve, to a large extent, horizontal chaining of activities. That is, the result which is produced by one activity, is fed to and consumed by the next. Such a horizontal binding of results is often represented explicitly by typed input/output objects associated to a particular activity. Usually, a set of conditions or criteria is attached to such input and output objects to determine whether their associated activities are ready to be executed or not.

**Explicit process representation** We have decided to represent process modeling components (i.e. task, product, tool, role, etc) explicitly. The immediate advantage of such decision is that process models are *tangible, persistent*, and thus ease communication and comprehension. An explicit process model can be easily manipulated, analyzed, and reasoned about. Moreover, such process models can be structured in such a way that it facilitates reuse.

**Tool integration** We don’t have intention nor sufficient resources to develop specific tools covering any particular life cycle activities. Rather, we will facilitate the integration of existing or new third-party tools. Such tool-integration facility is useful for end-user organizations when configuring their own process models. The decision to integrate third-party tools also give the flexibility and open-endedness of EPOS. To achieve that, we have to introduce formalism and semantics necessary to model a reasonable and sufficient part of actual tools. Some relevant issues which should be taken into consideration, are e.g. calling conventions, input/output objects, error message, etc.

**Process planning** or process model instantiation means to create required process instances (i.e. enactable process model variation) from a generic process description (e.g. types in EPOSDB) using suitable instantiation mechanisms. The process instances are also represented by entities and relationships in the EPOSDB, and generated by reasoning about a number of constraints given in process definitions, as well as the real world context in which the process is to operate. In particular, the EPOS Planner (see section 4.5.5.2) provides means to assist the user in this process creation. The process definition in EPOS is represented as entity and relation types which clearly need to extend beyond traditional, passive data types to describe their behavior nature.

**Dynamic and incremental planning** Software process is subjected to constant changes originating from different sources. It is thus important to be able to dynamically plan according to a specific circumstance. The EPOS Planner has been designed to support incremental planning, i.e. do not
plan until it is necessary. However, such an approach has also disadvantages, e.g. difficulty in having necessary level of detail at project start - hard to make comprehensive scheduling.

**Human and automated performance** Software engineering is a mixture of human-performed and automated activities. It is thus crucial to make the required distinction in order to provide appropriate support. That is, trivial tasks often involving pure tool invocation should be executed as soon as possibly, while tasks requiring human intervention must be initiated when it must suit the user. The EPOS Process Engine (see section 4.5.5.1) is therefore designed to have such support.

**Parallel and sequential execution** Activities in the process, both human or automated, generally have a *partial order*. Some can be executed in parallel, others only sequentially. Parallelism is fully supported in process instance (task network or plan) generated by the Planner. Since we have determined to have only one Process Engine for the entire task network, the execution parallelism is somehow limited.

**Flexible architecture for process model evolution** We apply late and dynamic binding of types to facilitate the immediate propagation of changes. That is, all EPOS process tools, including Planner and Process Engine, operate on information stored in the type at any time. It implies that any changes to the generic process model are immediately visible to its living enactable and enacting process models (so called top-down process model evolution).

The decision on integrating EPOSDB and EPOS-PM implies that EPOS-PM uses the same semantic data model (EPOS-OEER). A textual, simple modeling language (PML) was then defined to describe process models. Sequentially, we found this PML cumbersome, incomprehensible and thus hard to manage. We then define a new formalism for modeling processes, called SPELL (see more in section 4.5.3).

The decisions on adoption rule-based paradigm and required planning facility have strongly influenced our choice of implementation language. We have determined to use a free-ware SWI-Prolog and its graphical package XPCE for UNIX platform. This choice of Prolog complies to our need to have an environment where we can rapidly make prototypes, and it should not cost much to purchase. The entire volume of EPOS-PM, including kernel process models and associated supporting tools, is about 60K lines of Prolog code.

### 4.5.2 Modeling Concepts

The essential conceptual modeling notations in EPOS-PM are defined to represent the phenomena in the real-world software processes. They are:

**Task** representing activity or a logical unit of work, ranging from high-level tasks such as designing a system, to single tool invocations.

**Product** representing the software artifact as well as information about these. Product is consumed or developed by the tasks having them as input or output respectively.

**Tool** similar to Task except from that Tool do not require human intervention, and is encapsulated as an envelope with defined input/output parameters.
Person representing human beings, personnel or staff physically employed in an organization or explicitly assigned to a project.

Humanrole denoting the responsibilities and required skills of the persons performing the tasks.

Process models are described in term of these concepts and their inter-relationships, e.g.:

- Task – Product: A task has a number of input items, and outputs a set of new/changed items. The flow between tasks can be either direct or through a shared data store.
- Task – Role: A task is required to be carried out by a role which has suitable skill.
- Person – Role: A person with adequate knowledge and skill is assigned to play a specific role.
- Task – Task: Tasks may be related in several ways. There might be explicit constraints on the order in which tasks are enacted. Tasks may also be refined into a number of steps each being a task on its own.
- Product – Product: A product is related to represent the structure of a system, e.g. a system is consisted of several modules.

4.5.3 SPELL - Process Modeling Language

SPELL [CJM+’92], [NC94c] stands for Software Process Evolutionary Language. SPELL extends the DDL/DML of the underlying EPOSDB with behavioral object-orientation, explicit specification of inheritance rules, type- and instance-level properties, procedures and triggers. SPELL provides reflective language features which allow process models to be treated as process data and to be manipulated by a set of process tools. That is, evolution of process model can be described as part of the entire process. SPELL is expressed in Prolog which offers among others interpreting facility. Late and dynamic type binding are supported, and changes to process models (i.e. types) are thus immediately visible for all instances. SPELL supports several levels of abstraction and composition to model the external process elements including activities, tools, products, resource, projects and meta-processes. In this section, we initially present SPELL language constructs followed by SPELL inheritance and protection rules. We then describe the SPELL interpreter, its syntax and the high-level macros.

4.5.3.1 SPELL Language Construct

SPELL has extended the DDL/DML of the EPOSDB with behavior object orientation (i.e. instance- and type-level attribute and procedure), and reactive tasking framework (i.e. trigger). The new features offered by SPELL are described as follows (read together with the SPELL syntax in section 4.5.3.4 below).

instance- and type-level attribute specifying properties attached to a particular entity (type or instance). Instance- and type-level attributes are defined by a name, domain, and default value. Domain of attributes are one of those subtypes of AttributeDesc in figure 4.5.
instance- and type-level procedure representing methods or operations attached to a particular entity (type or instance). Defined procedures must be explicitly called or invoked. Procedures expressed in SPELL consists of two parts: an interface or declaration and a body. In the procedure interface it is necessary to define the procedure name, access rights and inheritance property. The access right has two possible values: private and public. The inheritance property includes three possible values: redef, append and inner. Detailed explanation of those values are presented in section 4.5.3.2. The body of the procedure is a sequence of actions expressed in Prolog.

trigger and rel_trigger similar to procedure, but its invocation is caused by a procedure call on a predefined condition. Trigger defined in SPELL has four main parts: i) proc - procedure name on which a given trigger is invoked; ii) when - is either before or after specifying whether the defined action part of the trigger is fired before or after the triggered procedure respectively; iii) cond - specifies on which condition of the object must be satisfied if the trigger will be fired; and iv) action - defines the trigger body. The trigger can be attached to either an entity or a relation type.

A SPELL example of type definition is shown in figure 4.7. The notation is introduced to facilitate easy understanding, and thus not syntactically correct. The example denotes the instance- and type-level attributes and procedures together with trigger definition.

ENTITY_TYPE XXX: XXX's Type {
  INSTANCE_LEVEL
    ATTRIBUTES
      Created : Boolean = ...;
      Status : String = ...;
    ...;
  PROCEDURES
    Upd_E(Self:OID, T:TID, [Attr-list]) = ...Created = false
    Qry_E(<Same as above>) = ...;
  TYPE_LEVEL
    ATTRIBUTES
      Condition : String = ...;
      Code : String = ...;
    ...;
  PROCEDURES
    Create_type(T:TID, [Attr-list]) = ...;
  TRIGGERS
    ON-PROC = Upd_E WHEN = AFTER
    COND = <If 'Condition' = ... > ACTION = <Carry out 'Code'>;
} %

Figure 4.7: A SPELL example of type definition

4.5.3.2 Inheritance and Protection

SPELL provides explicit specification of inheritance rules with dynamic binding for all type-level properties (attributes and procedures above). That is, when invoking a procedure, the code body, being ex-
executed, is synthesized by all procedure definitions in its supertypes according to defined inheritance rules. Three kinds of inheritance rules are available for type-level properties and for procedures:

**redef:** (as default) means overwriting or overloading the definition in the supertype.

**append:** logical conjunction.

**inner:** similar to Simula inner mechanism [DMN70]). It means, that if the code of a supertype T is defined as `step2`, and the code of a its subtype T1 is defined as `(step1, inner, step3)`, the executed code is turned out to be `(step1, step2, step3)` when procedure is invoked by subtype T1.

The protection or access right consists of two possible values: private and public. The private procedure can only be invoked by the type in which it is defined. On the other hand, the public procedure can be called by any arbitrary types. Similar notations have also been used in e.g. C++ to restrict visibility of data member and member functions of classes.

### 4.5.3.3 SPELL Interpreter

The access and bind semantics of procedures, triggers and of attributes according to inheritance rules are implemented by *one* Prolog predicate$^1$:

```prolog
call_proc(+Caller, +Callee, +ProcName, ?Result/Parameter)
```

where:

**Caller:** Type name or identifier (ID) of the caller. It is used to check access right defined in the procedure.

**Callee:** specifying the target the procedure is operated upon. It expects name or ID of the type and instance ID when accessing or invoking type-/instance-level attribute and procedure respectively.

**ProcName:** Name of the procedure. Some predefined procedures consist of `read, write, read_rel` are used to access both instance- and type-level attributes (see below).

**Result/Parameter:** the value returned after a query or parameter sent to procedure invocation. It has then a format like: `[Attr1-Val1, Attr2-Val2,...]`

When a type-level procedure is invoked, Callee is provided as input parameter instantiated to a type identifier, whereas for instance-level procedures, Callee is instantiated to an instance identifier. Three predefined procedures are defined for accessing (`read, read_rel`) and modification (write) both instance- and type-level attributes for entity and relation types. The `call_proc` has then following form:

```prolog
access: call_proc(Caller, Called, read, [AttrName-AttrValue])
        call_proc(Caller, Called, read_rel, [src/des, RelationId])

modification: call_proc(Caller, Called, write, [AttrName-NewAttrValue]).
```

$^1$For those Prolog predicates below (+) symbol in front a parameter denotes an mandatory instantiated parameter; (-) specifies variable returned while (?) can be either instantiated or uninstantiated variable. (_) is don't-care variable.
4.5.3.4 SPELL Syntax

Following is an example of a SPELL definition for an entity and a relation type in Prolog syntax. Note that words beginning with small letters is SPELL’s reserved words while those starting with capital letters are supposed to be replaced by a value.

entity_type{
    [name(TaskName),
     subtypeof(SupperTaskName),
     inst_attr([name-InstAttrName, domain-AttrDomain, default-Default]),
     type_attr([name-TypeAttrName, domain-AttrDomain, default-Default,
                inher-Inheritance]),
     inst_proc([head-InstProcName, access-Access, inher-Inheritance]),
     proc_body(head-InstProcName, body-Body),
     type_proc([head-TypeProcName, access-Access, inher-Inheritance]),
     proc_body(head-TypeProcName, body-Body)
    trigger([proc-ProcedureName, when-When,
             cond-Condition, action-Action]])}.

relation_type{
    [name(RelationName),
     subtypeof(SuperRelationName),
     rolecard(source-SourceRelation, LowCardinality-HighCardinality,
               dest-DestinationRelation, LowCardinality-HighCardinality),
     inst_attr([name-InstAttrName, domain-AttrDomain, default-Default]),
     type_attr([name-TypeAttrName, domain-AttrDomain, default-Default,
                inher-Inheritance]),
     rel_trigger([proc-Procedure, when-When,
                  cond-Condition, action-Action]])}.

Values of the name attribute for both entity and relation types must be unique possibly serving as a type identifiers. The name of the parent type must be specified for both entity and relation type.

4.5.3.5 Extending SPELL Syntax with High-level Macros

We have extended SPELL basic syntactical vocabulary with new features which make it easier to define process models. Those new syntactical features introduce high-level macros to make SPELL understandable and the interpretation more efficient. They can be summarized as following:

**Boolean expression** in PRE_DYNAMIC: It is possible to formulate the condition by any boolean expressions of predicates. It means that the condition in PRE_DYNAMIC can be expressed as a first-order predicate with arbitrary number of and and or keywords.

**Self-pointer** in definition for CODE, procedure and trigger body: It is highly convenient to refer to a given entity with a keyword self (similar to Simula self and this in C++).
Logical names for input/output parameters in EXECUTOR and CODE: This feature makes it easier to define the CODE-part without considering to use the correct identifier. Querying for a given object identifier usually appears to be very time-consuming and error-prone. Before executing the code, all the parameter macros are replaced by corresponding identifiers of actual inputs/outputs by obeying following rules:

- Parameter macros out\text{n} or in\text{n} - means a identifier of the \text{n}-th output or input respectively (\text{N} is an integer).

- Parameter macros outs or ins - represents a list of all identifier of outputs and inputs respectively.

4.5.4 Kernel Process Models

The EPOS process model is basically a typed network of chained and decomposed activity descriptions (tasks), expressing overall data flow and work breakdown respectively. The tasks are linked to descriptions of other tasks and/or products, tools, and roles. The tasks interact with each other, and with tools and humans. Since a task definition is also stored as a “product” in EPOSDB, certain (meta-)activities are allowed to work on such, e.g. for construction or execution of task networks. A Process Model in EPOS contains a set of types, where task types and associated triggers constitute the activation rules. Due to subtyping and (versioned) sub-projects, the EPOS type/rule base is basically hierarchically organized.

The PM-product is described by an object-oriented, “passive” ER data model, with procedures and triggers on entities (objects) to describe “active” behavior, and with explicit relationships. Technically, all process elements are modeled as persistent and versioned “products”, with special semantic extensions for activities and tools. EPOSDB (versioning, data modeling) and EPOS-PM go hand in hand: EPOSDB may exploit EPOS-PM to control and perform changes to evolving products. Inversely, the process model is itself a structured and evolving “product”, under EPOSDB control.

The hierarchy of predefined types for EPOS-PM is depicted in figure 4.8 and its part of it ER-schema is illustrated in figure 4.9. The layers presented in section 4.3 are also illustrated in the figure. Note that only a subset of kernel types of EPOSDB is shown in the figure.

As mentioned in 4.4.4, there are meta-types or type-descriptors in EPOSDB’s data model. SPELL (layer 2) extends EPOSDB’s kernel types with TaskDescriptor(TaskDesc) or DataDescriptor(DataDesc). All types are instances of either of them. In addition, new types, such as Procedure, Trigger, Rel_trigger, TypeAttribute and associated relation types are defined to represent the new language constructs of SPELL (see above). A set of new types are defined to support the tasking framework (layer 3), e.g. TaskEntity and DataEntity. They and their type descriptors are connected to represent work breakdown and chaining by new relation types, such as ActualParm, SubTask, FormalParm and DecompType. The root type for activity model (see 4.5.4.1) is TaskEntity while DataEntity is root of product model (see 4.5.4.3). A set of new types augmenting the types above is defined to describe the real-world processes (layer 5). They will be elaborated in the following sections. Figure 4.9 depicts a ER schema for the predefined types in figure 4.8.
4.5. EPOS-PM

Figure 4.8: Predefined Types in EPOS-PM

Figure 4.9: ER-schema for EPOS-PM
4.5.4.1 Activity (Task) Model

We distinguish between two main task subtypes: Interactor being an activity with human interaction (e.g. design, edit); and Deriver invoking a tool and executed automatically (e.g. compile, link - see also tool model below). A task type has many type-level attributes to control its static and dynamic behavior. Such type-level properties and their structure are declared in the meta-types, i.e. TaskDescriptor. The type-level task attributes serve as shared process parameters, and can be redefined by subtyping. Note that Prolog expression of the attribute are illustrated in parentheses. The type-level attributes are described as below:

**PRE_STATIC condition** (default-[]) represents the static conditions which are consisted of a set of required product states of inputs.

**POST_STATIC condition** (default-[]) represents the static conditions which include a set of required product states of outputs. PRE_STATIC and POST_STATIC conditions are evaluated by the Planner for task instantiation (see sec. 4.5.5.2). That is, they determine the horizontal chaining in the task network.

**PRE_DYNAMIC condition** (default-[], []) is two Prolog code sequences representing the dynamic condition of the task type. Such condition is evaluated by the Process Engine (see sec. 4.5.5.1) at run-time. The boolean result of such evaluation determines whether the task is ready to be executed or not. The PRE_DYNAMIC has two parts: temporal and non-temporal. A temporal part might be that a task should not execute before all its predecessors have terminated, or that it only should be compiled at night. A non-temporal part might be that the input of the task must have changed since the last execution of the task.

**CODE** (default-[], [], []) defines the sequence of actions performed when the task is executed, i.e. each time its PRE_DYNAMIC condition is evaluated to true. The CODE is written in an appropriate sequential language, p.t. Prolog. It often contains invocations to external tools. In that case, the task type serve as a tool envelope. The CODE has three parts: Pre-, Main- and Post-Code. The Main-Code part of high-level task must be empty. The Pre-Code is executed before the Main-Code invocation, while the Post-Code is mainly responsible for cleaning up or managing exception or assigning proper states for the outputs.

**DECOMPOSITION** (default-[]) defines the vertical task network breakdown and contains task type candidates of decomposed subtasks of a given high-level task. That is, DECOMPOSITION specifies the repertoire of valid task types possibly being its subtasks. A task with an empty DECOMPOSITION represents an atomic task, while a high-level or decomposable task has a non-empty DECOMPOSITION. Those high-level tasks initially issue a goal. Based on this particular goal a subplan or task network is generated and constructed. Relevant relation types for tasking framework (layer 4) are SubTasks which express the vertical decompositions, and GenInputs and GenOutputs which link a task instance to its actual input/output parameters being product descriptions.

**FORMALS** (default-[[in([])], [out([])]]) defines expected types for inputs and outputs. This type-level attribute is used together with PRE_ and POST_STATIC to construct horizontal chaining in task network. In other words, it is regarded as a task (tool) signature to constrain the product types of actual in/out-parameters. It is however possible to specify the dynamic occurrences of a particular input/output types.
STRATEGY has two possible value. goaldirected implies that the task is terminated after one execution. opportunistic indicates that the task will be executed as long as its PRE_DYNAMIC is evaluated to true.

STRICT is a boolean value. The true strict value for a task means that all its preceding tasks must be finished before the task can be executed. This type-level attribute is defined to facilitate both sequential and parallel task execution.

LEGALROLE constraints the execution of the task by limited number of roles.

An instance-level attribute TaskState specifies the current status of a task instance\(^2\). This information setting by the Process Engine (see section 4.5.5.1), has following possible values:

- Waiting: a newly created task will automatically be assigned this state. Waiting tasks are ready to have its PRE_DYNAMIC condition evaluated. If the evaluation results in failure the task state is remained. Otherwise, its state is changed to Active.

- Hot: similar to waiting-task state except from that task having this state, are more privileged to have its PRE_DYNAMIC condition evaluated. A task is assigned this state when at least one of its inputs is modified.

- Active: tasks are ready to have its CODE executed. After finishing the execution the task state becomes waiting or terminated state depending on the value specified in type-level attribute Strategy is opportunistic or goaldirected respectively.

- Forked: The CODE of the task includes an asynchronous invocation of an external tool and leaves control to this particular tool. The task is re-activated by a subsequent tool acknowledgment at completion.

- Terminated: final state of a task except the task is explicitly requested to be re-executed.

- Delegated: the particular task is delegated to another user working on another transaction but sharing the same product space. That is, the state of the task remains unchanged until that cooperative transaction is completed either by commit or abort.

Task execution is controlled by the TaskState attribute. A state transition diagram over possible values for TaskState is depicted in figure 4.10).

Below is an definition example of a high-level task type representing the Waterfall life cycle. The example is not presented in SPELL syntax, but a notation that makes it easy to understand for the reader.

In the example, we also include two user-defined instance-level procedures i_create and i_delete. Those two procedures are actually defined in the root node TaskEntity type, and are inherited and visible in all of its subtypes. Two other procedures, i.e. makegoal and subgoals are used by the Planner in building the task network (explained more details in section 4.5.5.2).

\(^2\)A task instance is from now on just referred as task.
Figure 4.10: Predefined task states and State transition diagram

4.5.4.2 Tool Model

Tools are modeled as task envelope connecting input/output parameters as signature. A tool is usually modeled by subtyping the predefined task Deriver attaching with an executable instance encapsulating around a logical tool name and formal parameters. The tool model has two following type-level attributes:

EXECUTOR (default-[toolname, parameters binding]) refers to the logical name of the tool that should execute the task. This type-level attribute explicitly specifies the name of external tools (e.g. cc, emacs, ...) which will be invoked and the parameter binding of its input and output.

SIGNSWITCH (default-[]) specifies the options which are included in the command to invoke the given tool (e.g. -o switch in c-compiler to re-direct binary output to a specific file).

An example of a C-Compiler tool type for C programming language is illustrated below:

4.5.4.3 Product Model

The DataEntity subtypes and corresponding relation types in Figure 4.8 express a part of our Product Model Schema. The type Part is used to describe any software part. A Family subtype describes a general collection of subsystems (a directory, a library), as well as a single module. A FamilyOf relation type, being a subtype of Composition, links together Familys constituting a hierarchical organization of families. Component, being another subtype of Part, represents atomic parts of
ENTITY TYPE Waterfall: TaskEntity {

    INSTANCE LEVEL

    ATTRIBUTES
    TaskState: := ...; % see figure 4.10.

    PROCEDURES
    i_create() = ...; % Constructor
    i_delete() = ...; % Destructor
    makegoal(Goal) = ...;
    % issue a goal for the Planner to
    % reason about in building subplan-sub-network

    TRIGGERS
    ON-PROC = i_delete WHEN = BEFORE
    COND = $self.TaskState = forked ACTION = some actions...

    TYPE LEVEL

    ATTRIBUTES
    PRE_STATIC (Inher=append) Pred = true
    PRE_DYNAMIC (Inher=append) Pred = true
    CODE (Inher=redef) Prog =＜empty＞
    POST_STATIC (Inher=append) Pred = true
    FORMALS (Inher=redef) String = 'in:$Requirement ⇝
    out:$ExecutableSystem'
    DECOMPOSITION (Inher=redef) String = 'REPERTOIRE(Analysis, Design, Implement, Test)'
    STRATEGY (Inher=redef) goaldirected
    LEGALROLE (Inher=redef) String = '<Logical role name, e.g. ProjectManager (see in 4.5.4.4).>'
    PROCEDURES
    subgoals(...) = ...;
    % used by Planner in sec. 4.5.5.2
    % to generate goal for each its inputs.
}

Figure 4.11: A Waterfall task type

software products. The component has two subtypes: Text denoting a primary object (e.g. document, source code) connecting to a Longfield in the EPOSDB that store the real content of the software parts; and Binary representing derived entities (e.g. object code, library). The product model has two following type-level attributes:

Legalsuffix (default-[]) specifies a allowable suffix for the file name to the instance of this type. E.g. .doc is usually a possible suffix for all files produced by text processor MS-Word.

Primary (default-[true]) is a boolean value determining whether instances of this type are either primary or derived objects. E.g. source code is a primary entity, while its generated object code is a derived object.

An example of C-File product type is depicted below:

In this example, we also illustrate examples of usage for triggers. There are two triggers attached to this C-file type. The first trigger is defined in such a way that responsible creator is notified if the compilation doesn’t succeed. Similarly, the second trigger changes the Taskstate attribute of all tasks which has an instance of this type after the execution of procedure edit.
ENTITY_TYPE C-Compiler: Deriver {

INSTANCE_LEVEL

ATTRIBUTES
TaskState: := ...; % see figure 4.10.
FORMALS (Inher=redef) String = 'in1:Sc-file ⇒
        out1:$object-file'
EXECUTOR (Inher=redef) String = '->cc, out1, in1:'
SIGNSWITCH (Inher=redef) String = '->-o>' % option -o for compilation.
}

Figure 4.12: A C-Compiler tool type

ENTITY_TYPE C-File: Text {

INSTANCE_LEVEL

ATTRIBUTES
Creator: := ...; % Name of creator.
Date: := ...; % Date to be created.

PROCEDURES
compile() = ...; % Compilation
edit() = ...; % Compilation

TRIGGERS
ON-PROC = compile WHEN = AFTER
COND = $self.status = failure ACTION = notify the responsible $self.Creator ...

TRIGGERS
ON-PROC = edit WHEN = AFTER
COND = $self.status = edited ACTION = tasks which has $self as input are set to hot task state.

TYPE_LEVEL

ATTRIBUTES
LegalSuffix: := c; % Legal file name
Primary: := true; % C-File is primary object

PROCEDURES
...
}

Figure 4.13: A C-File product type
4.5.4.4 Resource Model

We have two predefined types representing Person and Humanrole in this very simple resource model. Those two entity types are connected to each other by a relation type Employment. This relation type is designed to reflect the fact that one person can play several roles and a role can be occupied by several people. Another relevant relation type for the resource model is Assignment. This relation connects an instance of Humanrole type to an instance of TaskEntity and of course all its subtypes. That is, such relationships are created according to the information specified in the type-level attribute Legalrole in activity (task) model. When a task is to be executed, the role of the interactive person is verified before approving the execution.

4.5.4.5 Project Model

Project encapsulates a set of activities being executed to achieve an intended goal within a set of constraints (e.g., budget, time, resources, ...). Those activities are thus encapsulated by a EPOSDB transaction. Projects can therefore be broken down similarly in hierarchy as transactions. We have distinguished three different working environments or project predefined types (see also figure 4.8):

root_proj or a project dictionary owned by an executive manager being in charge of several independent, paralell projects in an organisation. The owner of this project is thus responsible for initiating several composite_proj (see below), and will always be informed about recent status of all its ongoing subprojects. Necessary information about the new project is required to be specified explicitly when initiating.

composite_proj or a project coordinator owned by a project manager working in a specific context limited by a set of constraints. In the same manner, composite_proj can manually initiate new atomic_proj as necessary by specifying necessary project information.

atomic_proj or a leaf-project doing real work and is owned by a process performer or developer. By using its pop-up menu Start working a high-level task can be created being the root for the entire task network.

A more detailed discussion on support for project initiation, tracking, and reporting is found in section 4.5.6.6 below.

4.5.4.6 Meta-process Model

Not only the product space is evolved as a result of change in requirements, but also the process used to develop the product needs to be updated according to new technology, thorough insight and gained experiences. Meta-process is thus responsible for managing and evolving processes in the same way as process does to product. Due to the fact that processes from the viewpoint of EPOSDB are products, our meta-process model which can be described by the same formalism in SPELL, is able to manipulate process models as its datum.

As mentioned above each type defined in figure 4.8 is an instance of an associated meta-type which is used to store type-level information. E.g. TaskEntity and DataEntity types are instances of
TaskDescriptor and DataDescriptor respectively. Thus, simple meta-activities can be defined in meta-types as instance-level procedures which describe how their instances (i.e. types from our point of view) are created and evolved in a controlled manner. There are three procedures to manipulate process schema (i.e. types or templates): t_create, t_delete, serving as meta-type’s constructor and destructor respectively, and t_change. The last procedure is of most importance when process models are evolved.

Such process model evolution certainly result in inconsistencies and discrepancies to process models. A type change may affect the extent of the type, i.e. the instances of the modified type as well as the related types of the modified type and their extents. The related types of a particular type are: its sub-types and types which are connected by type-level relations, e.g. FORMAL or DECOMPOSITION. We have therefore provide a set of basic mechanisms, constraints, and adequate support for analyzing potential impacts of a process model change. They are elaborated later in the discussion on the process tool Schema Manager in section 4.5.6.1. Changing to the meta-types themselves is performed by its type-level procedure mt_change. In short, the procedures defined in the meta-process model reflect the existing policy or strategy for changing all process models including meta-process model itself. However, such a meta-procedure can initiate a totally new task network consisting of several meta-activities (meta-tasks) whose operands (input/output) are process models. EPOS meta-tools, such as e.g. Schema Manager, Planner, Process Engine, are invoked by those procedures to instrument, build and evolve process models.

Below is an example of a TaskDescriptor meta-type, including procedures for meta-activities.

ENTITY_TYPE TaskDescriptor: EntityDescriptor {
  INSTANCE_LEVEL
  ATTRIBUTES
  PRE_STATIC % Structural
  PRE_DYNAMIC % declaration
  CODE
  POST_STATIC
  FORMALS
  DECOMPOSITION
  EXECUTOR
  SWITCH
  STRATEGY
  LEGALROLE
  PROCEDURES
    t_create() = invoke Schema manager; Planner % Constructor
    t_delete() = invoke Schema manager; Planner % Destructor
    t_change() = invoke Schema manager; Planner % Soft change (see 4.5.6.1)
  TYPE_LEVEL
  PROCEDURES
    mt_change() = invoke Schema manager
}

Figure 4.14: A TaskDescriptor meta-type
4.5.4.7 Consistency between Process Models

We discuss here a set of consistency constraints required to be met by different process models when interacting to each other. A process model is characterized as consistent if a set of type-invariants holds. They are:

- A type name must be unique over all versions. This is due to the fact that type name is also used to identify object in EPOSDB.

- A type must have only one super-type (single inheritance).

- Correct topological definition order: all supertypes or entity types occurring in relation types must exist before defining the given type.

- The type and all supertypes of an instance must be visible in the actual database version (i.e. sub-database associated to a particular EPOSDB transaction).

- Likewise for the visibility of the related entities of a relationship.

- The in-FORMALS parameters of a task type must be non-empty, and partially match the out-FORMALS of another task type.

- The tasks specified in the DECOMPOSITION must exist.

4.5.5 Basic Process Tools for Tasking Support

In the following, we present basic process tools providing support for the tasking framework. The Planner and Process Engine are considered as meta-tools since they are mainly operating on type information in EPOS-PM.

4.5.5.1 Process Engine

Process Engine schedules, interprets, executes, controls and record tasks. One process engine per transaction, cooperates with external tools and users. The Process Engine is responsible for invoking the Planner, and to execute and re-execute the plans generated by this. Such a plan is a task network, possibly partially ordered by the GenInputs and GenOutputs relationships.

The Process Engine checks and evaluates the PRE_DYNAMIC condition and determine whether a given task is ready to be executed or not. Each task type is associated with an execution Strategy attribute which has two possible values: goal-directed (lazy) or opportunistic (busy). The strategy can vary from task to task in the same task network. The goal-directed tasks remain Terminated until re-execution is explicitly requested. Opportunistic tasks are automatically reset to Waiting after CODE execution (see also figure 4.10).
4.5.5.2 Planner

The Planner [Liu90] works on a knowledge base consisting of static type information and a dynamic Product Structure as its World State Description. The reasoning of the Planner is based on the PRE_STATIC- and POST_STATIC conditions of task types. It applies hierarchical and non-linear planning.

Hierarchical planning is accomplished by coupling the Process Engine and the Planner. That is, when the Process Engine meets a task with empty CODE, it invokes the Planner to decompose this high-level parent task based on its DECOMPOSITION property. The Planner will take the current world state as the initial world state, the POST-condition of the parent task as the goal, and the task list in the DECOMPOSITION as the candidate subtask pool. The overall goal for parent task is issued by invoking its associated makegoal procedure. The Planner will then build a subplan to meet a overall based on the current World State. As new tasks are gradually added to the subplan, their individual goals are also integrated to the overall goal by invoking procedure subgoal. The generated subplan is added to the original plan through the parent's SubTasks relationship.

Non-linear planning is modeled as a production system similar to IPEM [AI87]. The generated plan will be a partially ordered task network, not a linear sequence of tasks. The Planner can thus deal with parallel processing and handle possible goal interactions. Guided by a proper set of domain-dependent heuristics, our non-linear planning is quite efficient.

Re-planning In our PM domain, task types have explicit inputs/outputs, or actual parameters. The Planner must therefore build these actual parameters, given the FORMALS description. This parameter building is described by extra generated assertions in the PRE_STATIC/POST-conditions. In [LC93], we have also described how the existing task network is re-planned due to changes to the process models or to the product structure.

4.5.6 Advanced Process Tools

In this section, we present advanced process process tools in layer 5 of figure 4.3. They are Schema Manager, Task Network Editor, CHAT, Transaction Planning Assistant and Project Manager.

4.5.6.1 Schema Manager

Schema Manager [GKS+92] meta-tool is responsible for browsing, editing, defining, analyzing, translating and evolving the Process Schema. The Schema Manager consists of:

Type Navigator gives possibility of browsing existing types as well as manipulating them. That is, a particular existing type can be deleted, modified; and a new type can be defined. The last manipulating functionality can be performed with assistance of The Type Editor.

Type Editor provides both textual and graphical user interfaces. The former one is strongly dependent on the user knowledge in SPELL while the latter tries to bridge such lacking gap by self-explainable and understanding notations.

Consistency Checker in both editing modus, it is feasible for the Type Editor to invoke the Consistency Checker to examine last updates. Modified parts of the Process Schema are both syntacti-
4.5. EPOS-PM

...ually and semantically checked for errors or inconsistencies according to the type-invariants defined in section 4.5.4.7.

Such a manner of defining and manipulating types is called top-down due to the fact that new process models and their interactions are basically built by the knowledge of existing type hierarchy. Figure 4.15 depicts the Type Navigator, Textual and Graphical Type Editor in the Schema Manager.

![Image]

Figure 4.15: Schema Manager illustrating by a Navigator, Textual and Graphical Editor

The possibility in process model customization and evolution in a hybrid machine-human environment is a crucial property for achieving sufficient flexibility. That is, classical programming language technology such as type-checking and compiled system are partly unsuitable for that purpose. On the contrary, interpretation, late type binding and reflexiveness are more adequate for dynamic behavior in most PM-systems. Below, we present basic mechanisms for managing process model evolution in EPOS, and impact analysis support provided by the Schema Manager.

In [JLC92] and [NC94b] we have discussed some basic mechanism on managing process model evolution in EPOS context. We distinguish two kinds of process model evolutions in EPOS:

**Hard or Structural** implies changes in instance-level property (attributes, procedure, trigger) or in the subtype structure. Such changes are generally disallowed by the EPOSDB, see however [Odb92] [Bra92] for more details.

**Soft or Behavioral** includes changes in type-level property. Such changes are allowed and manageable in EPOS.
There are three axes of evolving process models or soft changes in EPOS:

1. **Vertical** structuring by subtyping and other project specialization. i.e the activity rule base is *hierarchical*. That is, when there is a need for evolving a given type, a new subtype is created meeting all required changes. EPOSDB provides support for converting all *living* instances of the evolved type.

2. **Horizontal** evolution only happens inside projects. The new changes will be versioned according to the specified project ambition. As many of such evolution iterations occur, it is no doubt to witness to an explosion in the number of new *options*.

3. **Combined** vertical/horizontal, e.g. by change types on-the-fly without converting *living* instances. By adopting dynamic binding semantic the new changes will be visible to newly created instances.

The Schema Manager changes a type-level properties by using meta-procedure *t_change* in meta-process model. Changing types in EPOS may affect:

- The *living* instances of the modified type, or its subtypes.
- The subtypes of the modified type.
- The types otherwise related, like entity-types occurring in relation type or those connected by FORMALS/DECOMPOSITION.

We have extended the Schema Manager to provide features for dealing with such change impacts. We have still managed to address the change impact to the living instance. The living instances of a modified type is detected and following corrective actions can be initiated:

- **restart** if the change is applied to DYNAMIC_PRE, CODE, and EXECUTOR type-level properties, the procedure restart will cancel and undo the actions performed by task instance. Cancelling all the actions is impossible, since the CODE is allowed to contain whatever Prolog predicates. It is therefore supposed that when the user defines a task type, he/she also provides an instance-level procedure fail which is able to undo performed actions.

- **replan** if the change is done to STATIC_PRE, STATIC_POST, FORMAL, and DECOMPOSITION, the procedure replan will remove the existing subplan or task network of the modified parent task. The procedure will then invoke the Planner to build a new task network incorporating the new changes.

### 4.5.6.2 Task Network Editor

[Pass95] automatically creates textual SPELL definition or real types in EPOSDB from a user-drawn task network. User can draw a required task network in form of a workflow horizontally connected between tasks, and a work-division vertically broken down to smaller, less complex tasks. A user-friendly graphical user interface with a panel including kernel modeling components in EPOS-PM is provided. Kernel components consist of Task, Product, FormalIn, FormalOut, Decomposition, and Feedback. When a task network is completed drawing, the user can determine to build a process model
based on the graphical information. This type creation process must intelligently detect similarities and then enable reuse of existing types to the extreme extend. This kind of defining process model is termed bottom-up. New process model is built from the user's own perception of the process and its inherent flows illustrated by the tasking framework (horizontal chaining and vertical decomposition).

Figure 4.16: Screen picture of Task Network Editor

Figure 4.16 shows a screen picture of the Task Network Editor in EPOS. We have drawn a task network for develop task which takes a requirement document as input, and produces an executable file. The square represents a task while a product is depicted by an ellipse. Three new tasks (e.g. coding, review and compile) are then defined to be subtasks of the high-level develop task. They are also connected by other products. The review task is further decomposed to three other subtasks such as prepare, meeting and decide. The Task Network Editor can then generate SPELL definitions for those task and product to a textual file, or create types to EPOSDB. A picture of the task network in
Postscript format can also be generated.

4.5.6.3 Conflict Handling Toolkit (CHAT)

CHAT is an upgrading of our former Workspace Manager [Ngu93] which simply implemented check-in and check-out mechanism between EPOSDB and file-based workspaces. The check-out operation maps products into file structure to ease external tool access. A transaction is only connected to one workspace, and its *writets* and *readset* are defined to manage cooperation. Files specified in the *writets* are locked for modification by others unless there exists an agreed-upon cooperation protocol.

We develop the CHAT [Wan95] as we realize that the locking mechanism is too strict. CHAT is a system supporting more flexible cooperation for solving conflicts of sharing data which may occur. The system helps the user to avoid, to be aware of and to solve conflicts of sharing working data. CHAT automatically detects and notifies to users about potential conflicts between workspaces. The user has possibility in working in isolation and conflicting updates will be merged. In addition, the user is notified when a new version of working product is available in parent workspace, and the system can automatically check out the new version to the user’s private workspace. The users working on the same products can choose to gradually synchronize their temporary results between workspaces. CHAT also provides a graphical tool for viewing conflicting components, and then let involved agents to negotiate for the final version.

![CHAT - Conflict Handling Toolkit](image)

Figure 4.17: Screen picture of CHAT

Figure 4.17 depicts a screen picture of the main window of CHAT. From the picture, the user can see the name of the workspace (i.e. corresponds to a directory in the file system) he/she connects to, and the associated subsystems (i.e. correspond to sub-directories under the main one). The middle and
right windows respectively show the primary and derived components which belong to the selected subsystem in the window in the left. The font styles of the component names indicate their status. The component name with bold face is already checked out while the one with normal font is not checked out yet. A read-only component is displayed with italic font. Working with CHAT, the user is always notified about potential and actual conflicts which occur. The incoming notifications are displayed in the bottom window. The notifications which is printed with bold face, are critical and should thus be taken into consideration immediately. They concern e.g. a possible write/write conflict when another user has checked out the same product for update. In addition, the user can subscribe what kind of notifications he/she wants to receive during work. Such notifications are printed by italic fonts in the incoming message window.

4.5.6.4 Transaction Planning Assistant (TRAPAS)

In another work direction, an coordinating approach, called Transaction Planning Assistant [CLH95], has been proposed making the current cooperation support for long transactions more flexible. This approach relies on the Work-Unit description (WU in section 4.4.6.2), and describes a way to reduce the number of conflicts that may arise when several users cooperate to solve a task using a common database. Manual interaction between the users are made easier by helping a project manager to better partition, schedule and connect the users’ local activities as part of a common task.

Given a proposed project division, we can assess the connections between subprojects through impact analysis. Based on this analysis, the project manager can choose to adjust the initial partitioning to reduce the dependencies between subprojects. The impact analysis can further help the project manager to schedule the identified subprojects. Lastly, such analysis can be used to suggest suitable cooperation protocols between subprojects, being scheduled in parallel. Such protocols will guide mutual propagation and sharing of temporary results, and associated negotiation. Only algorithms and methods have been defined for TRAPAS. We have not implemented any tool support yet.

4.5.6.5 Broadcast Message Server

The internal EPOSDB message passing mechanism (section 4.4.7) is poor to satisfy the requiring demands in software engineering in which several external tools are invoked. This observation motivated for initiating a student project to implement BaMSE (Broadcast Message Server for EPOS) [ABrF93], whose objective is to introduce appropriate communication technology into EPOS environment. The chosen Broadcast Message Server (BMS) technology is HP SoftBench. The current and limited communication approach is improved by using BMS technology in which support is provided for effectively exchanging messages between multiple users distributed around a network and efficiently integrating with external tools. Specifically, a special filter tool against EPOS converts message formats and identity senders/receivers. However, such tools must consistently be described both as BMS table entries and as EPOS types.

4.5.6.6 Project Manager Tool

A set of simple project management practices have been designed and implemented in EPOS-PM. They include project initiation; task delegation; project tracking; and effort measurement. Those features
will be treated in following. This is a first step towards a full integration of project management practices in our EPOS system. However, the functionalities which are provided in the initial implementation of EPOS, is still limited.

**Project Initiation** From any project instances of type `root_proj` and `composite_proj`, it is possible to manually start or initiate a new `composite_proj` and `atomic_proj` respectively. Project information necessary for starting up a new project include project’s name, intention, ambition, choice, workspace directory, owner. The project name must be unique among ongoing projects. Intention is a textual description or information on what the new project plans to do. Appropriate option names using in ambition and choice are determined together with the specified workspace storing your developing products within a project. An email address is required for the owner attribute which will be used for communication purpose, e.g. sending him/her a message or notifier telling what to do. As soon as the new project is started, its project status is switched to `initiated`.

It is often that a project doesn’t possess adequate competence for performing certain tasks, and needs to delegate to other suitable projects. The support for enabling delegating particular tasks to another project is properly provided by the Project Manager process tool. The Task ID and the information about the project to which a given task is designated, must be explicitly specified. While the delegated task is executed somewhere else, the project can continue performing its remaining tasks. At completion of the delegated task, the project will be reported and results are available for further execution. Such a support is extremely valuable regarding the possibility for cooperating between different projects.

**Project Tracking** A simple status reporting routine from subproject to parent project is integrated in the Project Manager. Whenever the project changes its status due to external or human events, the new status is automatically reported to the parent project. Applying the intra-transaction communication (described in sec. 4.5.7.1) the project can at any time be notified about the status change of its subprojects. The project status changes can be caused by i) explicit project initiation; ii) the owner connects to the project, i.e. project is officially running; iii) The project owner finish or abort the project at commit.

**Effort Measurement** It is crucial in project management to keep control of amount of time used in each activity. Project Manager provides support for recording time usage for each task calculating from the time task is activated to it is completed. The time record is sent to a responsible monitoring task which keeps a textual record of all completed tasks.

### 4.5.7 Communication and Feedback in EPOS-PM

#### 4.5.7.1 Intra-transaction Communication

When executing task network, it requires a facility for sending various kind of messages back and forth between task instances. The EPOSDB message passing mechanisms (described in sec. 4.4.7) only supports for communication between EPOSDB transactions, not between tasks within a transaction. It implies that we implement a new feature, called intra-transaction communication. Each task instance is modeled to be attached by a Mailbox object which is able to receive a unlimited number of instances of predefined type `Message` or its subtypes. Basic communication primitives provided are:
• Testing mailbox for new incoming message. The true value is returned if there exists at least one message in the mailbox, and then activates the associated task for reading.

• Sending a message to a particular task by specifying TaskID. The communication is asynchronous. The sender task thus continues with other actions after transmision.

• Receiving message from the mailbox and making it available for further actions. Several messages can be read from the mailbox at one operation.

### 4.5.7.2 Feedback in EPOS-PM

When modeling several process examples, we realize a need for explicitly defining feedback loop between two tasks. We have therefore spent a great deal of effort addressing this problem. The Planner is modified to be able to build the task network including feedbacks. In addition, a feedback is defined as an integral part of the product space. The predefined Feedback type has following instance-level attributes:

- **revfile**: denoting the filename or identifier of the product to which feedback is intended. E.g. the attribute value for a review feedback is the document name being reviewed.

- **content**: describing the content of the feedback, either manually produced (e.g. review comments), or automatically generated (e.g. error message from compiler).

- **filename**: specifying the name of the feedback file. In some cases, people prefers to send feedback in form of a file rather than using content attribute above.

The feedback has two predefined subtypes (see figure 4.8. New subtypes can freely define to add necessary attributes or properties.

- **humanfb**: associating to human feedback or comment (e.g., comment of the reviewer or trouble report generated by the testers).

- **toolfb**: is derived automatically by invocation an external tool (e.g., compiler generates error messages).

In the following, a method to model feedback in EPOS are briefly described. For simplicity, a producer is the task which generates the feedback while a consumer is the task which receives the feedback. More detailed instructions can be found in appendix A. We distinguish two cases:

**Transmission Feedback at the same level**, i.e. the producer and consumer are resided at the same level in the task network. Assume that we want to model feedback toolfb generated by a compiler and sent back to a edit task. In the producer task type there are two issues which must be taken into consideration:

1. the STATIC_POST for feedback in the producer task should be defined to be none.
2. the new type-level procedure toolfb_receiver must be defined. This procedure is used to determine the input filename of task which the feedback will be sent to (i.e., the edit task)
In the consumer task type following issues must be taken care of:

1. **STATIC_PRE** for feedback is specified to be in appropriate state (e.g., initiated).
2. the new type-level procedure `subgoal_edit` must be defined to issue goal associated to the feedback input to the Planner.

**Transmission Feedback at different levels** occurs when a feedback is generated and consumed at different levels in the task network. The particular feedback is kept moving up to higher level (i.e., to its parent tasks) until its transmission is taken place at the same level in the task network. The method described above is then applied.

### 4.6 Claimed Contribution

In the following, we clarify contribution of the author which is divided into three categories. The publications of the author’s contribution are also explicitly noted.

#### 4.6.1 Collaborative contribution

- Design and implement EPOS-PM’s modeling concepts (section 4.5.2) and process modelling language SPELL including language syntax, construct and its interpreter (section 4.5.3). The contribution of the author is published in [CJM+92] and [NC94c].

- Design and model task, product, tool, resource, project and meta-process models (section 4.5.4) in EPOS-PM. Such kernel process models (i.e. types) constitute about 1200 lines of Prolog code.

- Introduce and elaborate concepts of meta-process and their associated meta-activities both in general focus and specifically in EPOS [NC94b]. Simple procedures for evolving process models in EPOS are also implemented.

#### 4.6.2 Own contribution

- Design and implement support for managing cooperative work within the context of EPOS-PM ([NC93] and partly used in ISPW7 modeling example in section 5.4). The issue of cooperation support is now described in section 4.4.6. This work has resulted in about 1800 lines of Prolog code.

- Develop a workspace manager [Ngu93] prototype which supports check in/out product to/from the EPOSDB. It also manipulates workspace layout to support cooperative transaction (in section 4.4.6. The workspace manager prototype and its ideas are considerably improved and implemented in CHAT (section 4.5.6.3).

- Introduce high-level macros in SPELL to facilitate modeling processes (section 4.5.3.5).

- Design and implement Project Manager tool (section 4.5.6.6). They include project initiation, delegation, tracking and collection of effort measure. The project model and Project Manager tool are implemented by about 1700 lines of Prolog code.
4.7. Temporary Evaluation of EPOS baseline

- Implement intra-transaction communication (sec. 4.5.7.1) and feedback mechanisms (sec. 4.5.7.2). The Planner and the Process Engine are upgraded accordingly. This work is realized by about 1000 lines of Prolog code.

- Upgrade and maintain the entire EPOS-PM system covering from process models to SPELL and various process tools (i.e. Process Engine, Planner and Schema Manager). The modifications are made to incorporate new features when experimenting with process examples (see chapter 5).

4.6.3 Supervising work

- Supervise a student group to implement the Schema Manager prototype [GKS+92]. This version has been upgraded by another student and the author.

- Supervise a student group to implement BMS (Broadcast Message Server for EPOS) [ABrF+93].

- Supervise a student to implement Task Network Editor [Pas95], and maintain this work afterward.

4.7 Temporary Evaluation of EPOS baseline

Below, we evaluate our current work against the requirements stated in chapter 3. A complete evaluation is postponed to section 10.4 in concluding chapter 10.

- We have introduced a set of modeling concepts to facilitate modeling phenomena within real-world software processes. They include task, product, tool, person and humanrole. Those modeling concepts are consistently related and familiar to any body. A process modeling language SPELL is also defined to express process models. SPELL includes formal notation to represent both structural and behavioral properties of software processes. Due to the object-oriented property of SPELL, it provides both abstraction and modularization of the process models. The SPELL written syntax is however slightly hard to understand. We have therefore designed and implemented tool support which offer graphical notations, to facilitate the modeling task (e.g. Schema Manager, Task Network Editor). The instantiation of process model is automatically and incrementally performed by the Planner. The Planner constructs the enactable process model from the generic one (i.e. types). The Process Engine which is associated to an enacting process model, interpret and enact the software processes. It provides automatic execution of trivial task and allows interactions with human being when enacting the process model.

- The data management issue is supported by the EPOSDB which offers configuration and version management of both software products and software process models (i.e. types). The nested and long transaction in EPOSDB facilitates for effective concurrency control and coordination in engineering work. Flexible cooperation protocol and tool support (CHAT) enable sophisticated communication and collaboration between EPOSDB transactions. The modeling concepts in EPOS-PM is still limited to support modeling a real software organization including several organizational units and parallelly ongoing project context. The Project Manager Tool simply provides features to initiate, delegate, track and terminate projects. Support to revise project parameters before and during project execution is still missing.
• Many PSEEs as well as EPOS have spent a great deal of effort to introduce methods and mechanisms to effectively managing generic process model evolution, and then propagate to corresponding enactable and enacting process models (top-down process model evolution). However, our hypothesis is that such changes only constitute a marginal part of the total number of changes during project. There are a larger number of changes to the initial project plan or the task network. We need to better understand those changes before we can elaborate approaches to deal with that. Our hypothesis must also be validated by actually collecting changes during real software projects, and our new approaches can then be verified.

4.7.1 Further Ideas for EPOS Extension

Based on the temporary evaluation of initial implementation of EPOS above, we outline further work directions for the remainder of the thesis.

Modeling experience When the necessary features for process support in a PSEE in place, it raises a need for validating them by modeling some concrete process examples. The gained modeling experiences will certainly motivate for further enhancement and improvement. The modeling experiences with EPOS are reported in chapter 5.

Understanding of process evolution There is still no attempt to understand the properties of process evolution in the current implementation of EPOS. Therefore, we need to investigate and elaborate suitable means to better understand of the nature of process evolution and gain insight into how it relates to external project characteristics. Such work is reported in chapter 6.

Managing process evolution In the current EPOS implementation, the evolution of process models is addressed by the modifying the generic process model and propagating changes to enactable and enacting process instances. Such evolution is referred to as top-down process model evolution. We are still interested in having facilities to change to process instances without affecting the generic process models - so called bottom-up process model evolution. Such extention of features in EPOS is presented in chapters 7 and chapter 8.

4.8 Chapter Summary

In this chapter, we have given a detailed description of architectural issue as well as main features provided by EPOS. Along to the presentation of the design and implementation of EPOS, we also outline the underlying design decisions which are made. EPOS is a multi-user, software engineering environment. EPOS supports integrated software configuration and process management through a two layer architecture: 1) EPOSDB - providing versioning/configuration management, object-oriented ER data model, and long, nested and cooperative transaction models; 2) EPOS-PM - including a set of predefined process models expressed in a reflexive, object-oriented software process modeling language SPELL. The software process models, representing activity, tool, product, resource, project and meta-process, are built, executed, and evolved by a set of process tools.

Finally, we conduct a temporary evaluation of the EPOS baseline with respect to the stated requirements. Such evaluation is useful to identify our current research position, shortcomings, and then further work directions for the remainder of the thesis.
Chapter 5

Modeling Experiences with EPOS

5.1 Introduction

In this chapter we describe our practical experiences done with EPOS-PM. The experiences gained has both validated our current work and motivated for further research. Our validating work essentially focus on modeling as many as possible process examples which are both practical and well-known to process community. By doing so, we enable opportunity for discussing, comparing and then considerably improving our prototype. Our first process example is a “toy” internally-defined development process aiming at simulating the waterfall life-cycle (section 5.2). Two more familiar process examples ISPW6/ISPW7 are presented in the following two sections. Finally, we model an industrial review process which actually performs at a local software house. The primary objective of this work is to thoroughly validate modeling, planning and enactment features provided by EPOS.

5.2 Modeling a simple Waterfall Life Cycle

Our first trial with the modeling facility of EPOS-PM is a “toy” process example in which a simple program system is developed. The program system includes a set of modules written in C-language within an UNIX environment. The process model is described to cover main phases in waterfall life cycle. Extended task and product types and a screen picture of the task network are depicted in figures 5.1 and 5.2. The new process models (i.e. types) is about 1500 lines of Prolog code.

Figure 5.2 depicts an actual graphical user interface of EPOS displaying a concrete task network in which an executable system \texttt{m.exe} is developed according to the change order \texttt{f.cho}. As the process model strictly follows the waterfall life cycle, the high-level develop task is decomposed to sub-tasks representing all development phases, such as design, review, implement and test. By performing the design and review tasks a product structure which is represented by a family \texttt{f}, is determined. As a reminder, a product structure is a hierarchy of tangible products which are inter-related and connected by a set of relationships (e.g. depends\_on and implemented\_by in figure 5.1). The implement task is then elaborated in two main activities, such as systemcoding including editing work of all involved bodies and interfaces; and build consisting compilation and linking developed parts into the final executable system. A root of the instantiated product structure is represented by a family \texttt{f} which contains two sub-families \texttt{fa} and \texttt{fb}. Developing of those three families are represented by the high-level systemcoding tasks which are decomposed to leaf-node editing tasks, called sourcecoding for each of their components. The inputs for each sourcecoding are an inter-
face (.h), code body (.c) and a document (.tex). The build process simply consists of compiler toolcc; archiver toolar; and linker toolld. Those tools which are subtypes of task deriver, are explicitly bound to external tools provided by UNIX, and automatically invoked as soon as the corresponding sourcecoding, or inputs (bound interface and body) are available. The test task is simply modeled to run the final system m.exe and inquire the approval of the user.

By modeling and enacting this process example we have managed to rate following feature as satisfactory: i) the power of expressiveness of process formalism; ii) the enactment facility; and iii) the planning feature. However, there exists no iteration in the process model. It means that errors detected by the build process are not taken into consideration and thus no corrective actions or feedback loops are initiated. Such a feature is trying to be addressed in the next modeling examples.

5.3 Modeling ISPW6 Example

The ISPW6 process example [KFF+90] has been designed to aid in understanding and comparing approached to software process modeling. By doing this, the process community can evaluate the relative strengths and weaknesses of the approaches under examination. The core problem of the example is focused on localized change to software system. This is prompted by a change in requirements, and can be thought of as occurring either late in the development phase or during the support (maintenance and enhancements) phases of the life-cycle. The problem context is confined by assuming that the change request has already been analyzed and approved by appropriate authority (e.g. Configuration Control Board). It has also been determined that only a single code unit (e.g. a module) is to be affected. The problem begins with the project manager scheduling the change and assigning the work to appropriate
Figure 5.2: Example of simple waterfall life cycle in EPOS
personnel. The example problem ends when the new version of the code has successfully passed the new unit tests.

The original solution of example ISPW6 has been presented in [CLW90]. We have improved the solution and actually implemented in the EPOS meeting the initial requirements of the example problem. Figures 5.3 and 5.4 depict the extended types and the task network respectively. The ISPW6 process model is about 1600 lines of Prolog code.

![Task Model Diagram]

![Product Model Diagram]

**Figure 5.3: ISPW6 Extended Types**

The task network in figure 5.4 addresses the issue of feedback between various tasks located in different levels. An example of feedback at the same level is illustrated by fb\_design connecting between output of review and input of modify\_design tasks. In contrast, fb\_edit connects between test\_unit and modify\_code residing at different levels in the task network. In addition, the EPOS intra-transaction communication by means of mailbox is also exploited to meet the example requirement on reporting and monitoring activity progress and effort. Specifically, the monitor\_prog task receives notifications and report on effort being spent to complete all technical activities. The collected effort measures are recorded in the document called proj\_log for comparison and assessment.
with respect to the initial project plan (proj_plan). In this example, we have also found necessary to exploit the boolean expressions (AND, OR) in specifying PRE_DYNAMIC conditions to satisfy the required control flow in the process description. The product model necessary to model ISPW6 is more complex than the previous example, including also new feedback types.

The lesson learned by performing this work is, except from re-validating the three facilities in 5.2, visualize the usefulness and necessity of having capability to express feedback, and intra-transaction communication in process models. A comprehensive guide to run the solution of ISPW6 problem can be found in the appendix A. In this guidance, we describe how to initiate a project context in which the example is performed. In addition, we also present how to delegate certain activities to another person connected to the same network. That is, facilities involving project management are validated.

5.4 Modeling ISPW7 Example

The ISPW7 example problem is promoted as part of the Seventh International Software Process Workshop (ISPW-7) and the gained success of the previous process example ISPW6. Rather than trying to construct a completely new example, the ISPW-7 example extends the original ISPW-6 example. The extensions attempt to address issues in two specific areas: teamwork and process change. The extensions address two specific issues in teamwork. One issue is the managerial problem (resource management) of allocating various constrained resources, especially human resources, in order to complete a schedule of tasks. A second issue is the coordination of multiple programmers working in parallel. The process change extensions addresses two forms of change. One, which we term “process modification,” is intended to represent some form of permanent evolution of a process. The second form of process change, which we term “process exception,” concerns a temporary modification to the process to handle some exceptional circumstance.

In [CAJ+91] we have proposed an original solution of the example problem mainly focusing on the teamwork aspect, and specially in communication and coordination area. As we have spent more effort after submission on cooperation issue [NC93] and workspace management in parallel software development [Ng93], a more complete and satisfactory solution is depicted in figure 5.5.

In figure 5.5 above there are two main tasks below high-level working task, i.e. cooperate task taking care of detection and reconciliation of conflicts, and develop task mainly handling the production process which can either be the waterfall or ISPW6 examples above. The working task is started within a EPOSDB transaction context, and can possibly be detected as being conflicting with one or several running overlapping transactions. In such a case, a cooperation protocol is determined by negotiating between transaction owners and represented by protocol. Affected components by such a conflict in the product structure are organized in a "pool", called watchstruc for easily resolving possibly conflicting updates. The communication between overlapping transactions are provided by the EPOSDB. The cooperation bobble in figure 5.5 is blown up and illustrated in figure 5.6. The process model for ISPW7 is about 1800 lines of Prolog code.

The key idea of modeling support for cooperation consists of two main activity flows:

**Outgoing:** includes three tasks auto_update, propagate_out and request. The first one monitors and detects local updates to the relevant product components in watchstruc. As a consequence of such a detection of product update, the second task composes a message and
Figure 5.4: ISPW6 task network
Figure 5.5: ISPW7 solution in EPOS

Figure 5.6: The cooperation transaction task network
sends it to all cooperative transactions notifying about the corresponding changes. The third task concerns about other kinds of inquires needed to communicate with cooperating transactions, such as wish to re-negotiate the cooperation protocol; notify about the pre-commit and etc. …

**Incoming:** mainly consists of `propagate in` task which receives incoming messages from other transactions and then dispatch them to appropriate tasks based on their types. Possible actions can be invoked and modeled in three different tasks: `merge`, `handle_commit`, and `re-negotiate` based upon what kind of propagation policy is determined in the cooperation protocol. Change propagation in `eager` modus activates `merge` task while change message from cooperating transaction with `lazy` propagation modus is sent to `handle_commit` task for reconciliation. `re-negotiate` task deals with message requiring modification of the initial cooperation protocol.

Modeling ISPW7 example has validated the ability in EPOS to communicate across transactions aiming at resolving conflicting updates when products are shared. It has also shown that EPOS formalism is sufficiently powerful to express an adequate cooperation apparatus associating with an arbitrary protocol. The availability of intra-transaction communication has once again demonstrated its capability in synchronizing and invoking appropriate actions regarding to the type of incoming message.

### 5.5 Modeling an Industrial Review Process Example

As a need for validating EPOS modeling capability with real software process is identified, we seek for a published industrial software process. Fortunately, we have through a close contact with a local software provider, got a process description on their internal review process.

The review process is initiated by an author that wants to have one or more of his documents (i.e. `test.doc`) reviewed by other project participants. Several authors may decide to have their documents reviewed in one process, one of the authors then initiate the review process.

1. **Create CFR** The author creates a review report (Call For Review - `test.cfr`). He fills in the name of the review manager, review secretary, and reviewers. He then writes the scope of the review, and determines time and place for the review meeting. He ensures that every document is set to `draft`, except `CFR` that maintains `under_work` status. He then distributes the review report and the associated documents to all reviewers.

2. **Review meeting** Decomposed. The review takes place and results documented in the review report (CFR). The updated CFR is printed and its status is unchanged (`under_work`).

2.1. **Organize** The reviewers read and give comments to the documents. If reviewers are not able to participate in the review meeting they deliver their comments to the review manager.

2.2. **Write protocol** The reviewers agree on a set of comments on all documents, these comments are documented by the secretary in the review protocol (considered to be a part of the CFR). Each document is given a review result, being either `Ok`, `NotOk`, or `Corrected`.

2.3. **Allocate controller** If the review result is determined to be `Corrected`, a controller will be allocated to verify the changes to be done later.
3. **Set document attribute**  The meeting manager changes the status of the document with review result. The document status is changed from under\_work to either Ok, NotOk or Corrected. If the document has status NotOk, the author is notified by update\_fb and has to update his document which will be reviewed again by starting from step 1 above.

4. **Change and control**  Decomposed. This activity is only performed if the document status is assigned to be Corrected. The process produce a new version of the document with status set to draft.

4.1. **Change document**  The author retrieves his document and updates it according to the CFR document with review comments. The review meeting has decided that the new changes in the document will be controlled by the allocated controller. The updated document is printed and sent to the controller.

4.2. **Control document**  The controller checks the updated documents according to the CFR. If the new changes are not approved, the comments (change\_fb) are sent back to the author who must modify his document again until the controller is satisfied.

5. **Approve CFR**  The CFR is set to approved by the project manager.

6. **Approve document**  If the document status is set to Ok after step 3, this activity is performed to complete the review process.

The task network of review example is depicted in figure 5.7. The document test\_doc is reviewed, and review comments are recorded in the test\_cfr (i.e. CFR). The notification about rejection is modeled by the feedback update\_fb from set\_attribute task to update\_doc task. The feedback from the controller, in case the document is assigned status Corrected, is implemented by the change\_fb which connects the control and change task together. The review process model is about 1200 lines of Prolog code.

By doing this modeling example we have strengthened our faith in our work by recording the fact that is is possible to model, enact and assist a real industrial process. Through the review process example we have also learned that the real industrial process is more complex, involves complicated iterations, and need substantially more support than other examples we have conducted. However, the requirement on explicitly assigning, notifying reviewer candidates is not satisfactorily met due to technical restrictions in EPOS.

### 5.6 Assessment of Gained Experiences

The experiences gained by modeling software processes mentioned above has provided us useful insights:

- Our modeling concepts satisfactorily describe software processes in the examples. EPOS supports for instantiation and enactment of process models have also been proven to be adequate by real execution of the models within the EPOS context.

- EPOS capability of managing process model evolution has not been clearly demonstrated by those selected modeled processes. We have to constantly change the generic process model variations during implementation of the examples. We did change directly to the generic process
models without suspending the ongoing execution of the affected task network. After changes are properly installed, the task network is executed according to the new revised generic process model.

- Although the ability to configure and enact the process plan (i.e. task network) is necessary to guide and enforce process performance, we perceive that it is also important to be capable of effectively managing and adapt any kind of unanticipated changes to the task network during project execution.

## 5.7 Chapter Summary

In this chapter, we have presented our modeling experiences with EPOS. The modeling examples are: a waterfall development life cycle, ISPW6, ISPW7, and a review process from a software company. Experiences and lessons gained by applying EPOS in modeling both process examples and industrial process are also described and constitute inspiring factor for further research.

The author has been the main actor in designing and modeling most process examples described in this chapter. The author contribution in modeling experiences with EPOS can be outlined as follows:

- Design and implement the Waterfall life cycle (section 5.2) for validating EPOS-PM.
- Design initial solution for ISPW7 [CAJ+91] in EPOS addressing the issue of cooperation.
• Re-design and re-implement original ISPW6/ISPW7 solutions in EPOS to better meet the requirements, and actually model and enact the process models within EPOS context (sections 5.3 and 5.4).

• Design and implement an example of review process performed at an industrial software provider (section 5.5).
Chapter 6

Categorization Framework for Evolution

6.1 Introduction

In this chapter, we will present a categorization framework for process evolution and a characterization framework for software project. They will then be used to relate properties of an evolution occurrence to typical, external project characteristics to better predict, anticipate and manage process evolution. The chapter is organized as follows. Section 6.2 presents some basic decisions or underlying thoughts for elaborating the categorization framework for process evolution. Section 6.3 presents basic structure of the categorization framework and its properties. Section 6.3.4 illustrates an example of how an operational framework is constructed. The project characterization framework is described in section 6.4 followed by a discussion on the relation between external project characteristics and process evolution in section 6.5.

6.2 Basic Decisions

The evolutionary nature of both software and associated software process has been under study for some time [LB85]. Such process evolution is hard and costly to manage and predict during its execution. Frequent changes are characterized as a major cause for late deliveries, cost overruns, missing features, and generally poor quality. We have in earlier chapters recognized a need to better understand and thus manage process evolution during software projects. In this chapter, we therefore present a framework which is used to categorize process evolution occurrences. As none of these evolution occurrences are alike, it is necessary to elaborate an appropriate categorization framework to easily group them in relevant patterns. It is then straightforward to evaluate adequate corrective as well as preventive initiatives to deal with them. Such initiatives are e.g. infusion of new development technology, introduction of software measurement or improvement of development process. In general, it is important to fully understand the problems before they can effectively be managed. A constant requirement revision can e.g. be identified as one of frequent process evolution patterns. Such unstable requirements increase the risks to meet project schedule. Developers can not design or code at a moving target. Changes which are occurred later in the life cycle have major ripple effects that may impact both product and schedule risk. For instance, the effort for defect removal is two times higher after subsequent phase.

We also believe that it is not realistic to talk about evolution patterns for software projects in general, without linking them to different project types. That is, information of project in which process evo-
olution occurrences are detected, must also be recorded. We can then easily reveal important relations between them. On the basis of such relations, we can anticipate the impact of evolution patterns which likely occur in a given project type. It means that project is better prepared for and can thus effectively react to process evolution. The ability to handle such evolution is a characteristic feature of successful projects. Therefore, we will in this chapter also present a framework to characterize project and link it to process evolution history.

6.3 Categorization Framework for Process Evolution

In this section, we first describe the basic structure of the framework followed by an outline of its typical characteristics. An approach to construct a categorization framework for a particular situation is presented, and then illustrated by an example. The fully constructed framework is used in the case study which is presented later in chapter 9.

6.3.1 Basic Structure of the Framework

The categorization framework for process evolution has three-level hierarchical structure. It contains three major concepts (dimension, aspect and category) and their interactions. Figure 6.1 depicts such structure of the framework. In the following, we describe its concepts in details.

Dimension represents a most abstract and top level of the categorization framework for process evolution. It includes pertinent questions relating to process evolution such as

- Where a process evolution is originated (i.e. evolution origin);
- Why a process evolution is caused (i.e. evolution cause);
- What is affected by a process evolution (i.e. evolution scope);
- When a process evolution is detected or corrected (i.e. evolution time);
- How a process evolution is handled, i.e. which impacts a process evolution implies (i.e. evolution reaction);
- By whom a process evolution is corrected (i.e. evolution corrector).

A particular process evolution occurrence is categorized by providing answers to those above questions within a given domain. A domain can e.g. be a software development or maintenance process. However, such answers can be of general nature and thus hard to be formally analyzed and reasoned about. A specification of such answers is therefore necessary. This is done by introducing two further levels (i.e. aspect and category) downward the framework hierarchy.

Aspect serves as a further specification of a given dimension. The aspects are determined by elaborating along axes reflecting different concerns or viewpoints of the dimension. A process evolution occurrence can be perceived from different viewpoints. Alternatively, a process evolution occurrence can result in implications reflecting different concerns. The aspect level is introduced to facilitate abstraction of the large numbers of possible answers to questions in the dimension. Still, they are not sufficiently concrete to extensively describe a particular evolution occurrence.
6.3. Categorization Framework for Process Evolution

**Category** constitutes the non-decomposable leaf node of the framework, and associates to one or several aspects. Categories are descriptive, and fully characterize any process evolution occurrence. The necessary categories are defined according to the particular situation in each organization. That is, categories are context-dependent and should be elaborated accordingly. In that way, categories appear to be familiar to the users of the framework. The number of defined categories should be kept optimal, i.e. both large enough to represent arbitrary process evolution, and minimal as well to be maintainable and to meet our needs for understanding the nature of process evolution.

![Diagram of Categorization Framework](image)

Figure 6.1: Basic Structure of the Categorization Framework

To categorize process evolution, the framework basic structure must be constructed according to a specific context (more detail in 6.3.3 below). A complete categorization framework is constructed by vertical specification (i.e. dimension - aspect - category) and horizontal elaboration (i.e. evaluation of concepts' parameters). In other word, appropriate aspects and categories are determined according to a given question within a particular domain and their parameters (i.e. concern, viewpoint and context).

### 6.3.2 Framework Characteristics

In the following, we discuss the basic characteristics of the categorization framework for process evolution. The categorization framework is:

**Explicit** The categorization framework shows the basic concepts (i.e. dimension, aspect and category) and their interactions of process evolution. Such a characteristic is useful to separate different concerns, and direct focus of research to address the categorization issue of process evolution during software projects. An explicit framework is also required if it is compared to another frameworks which address the same categorization issue. Moreover, an explicit framework conveys clear and direct message.

**Evolved** The categorization framework can be initially constructed and incrementally evolved. As process evolution occurrences are captured during software projects, new aspects and categories
can easily be incorporated into the framework. It means that the framework can dynamically be re-configured along the elaboration direction. Such feature is crucial to mature the initial framework as absolute all relevant information can not be determined at one time. That is, the framework usage and improvement activities are interleaved with each other.

**Continuous** The categorization framework can be continuously used during software projects. At project start, the framework can be used to predict and anticipate process evolution based on the already-known and external project indicator/profile (c.f. project characteristics in section 6.4). During the project execution, process evolution occurrences are categorized and recorded according to the framework. In addition, a specific, recorded process evolution occurrence can be retrieved by assigning values to the framework concepts. Such retrieval of previous similar process evolution is substantial to estimate impact or consequence, and then determine adequate actions of the new one during project execution. At project completion, the framework is then used to extract typical and frequent evolution patterns and their associated impacts. Such experiences found a quantitative basis for more effective planning and performing new future projects.

**Abstract** The categorization framework is organized hierarchically. Such a structure effectively hides a detailed, context-dependent categories by higher level and context-independent concepts such as aspects and dimensions. Such abstraction feature is also useful when we want to restrict the retrieval of previous process evolution occurrences to only aspect-level. E.g. we are only interested in investigating the frequency of process evolution occurrences associated to aspect \( A_1 \) in where-dimension, rather than frequencies of all its associated categories.

**Comprehensive** The categorization framework increases human understanding by initially asking general questions in the dimension, and then gradually specifying and elaborating them. The identification of aspects and categories in the constructed framework are made from a given context so that they should be familiar for the users.

### 6.3.3 Constructing a Categorization Framework

We present below an approach to construct a concrete categorization framework based on the basic structure above. The vertical specification process is described by proceeding the three steps below. On the other hand, the horizontal elaboration processes are described in step 2 and 3 for aspect and category respectively. An example of such construction process is illustrated in section 6.3.4.

**Step 1.** The intention of using the categorization framework for process evolution must be explicitly specified. On that basis, a minimal set of the identified questions (where, why, what, when, how and by-whom) must be selected to meet the intention. It is thus not necessary to include all those six questions. The domain in which the categorization framework is used, must also specified.

**Step 2.** The selected questions are elaborated in either viewpoint or concern axis. The choice of appropriate axis depends on specific question in the dimension. If the question is further elaborated along the viewpoint axis, we have to determine all aspects which are perceived by different roles of interest in the organization. On the other hand, different aspects are defined to reflect relevant concerns of process evolution in the question. The result of this step is a set of aspects which is further specified by context-dependent categories in the next level. The horizontal elaboration process for aspect is expressed in pseudo code as following:
6.3. Categorization Framework for Process Evolution

For each question in dimension do
    If viewpoint axis is selected then
        For each role of interest do
            determine aspects which are mainly perceived by the given role
    Else if concern axis is selected then
        For each concern of interest do
            determine aspects which best reflect the given concern

Step 3. For each aspect, a set of relevant categories must be determined according to the specific organizational context. That is, appropriate categories which are familiar to the organization at hand, are assigned to each aspect identified above in step 2.

6.3.4 An Example of Constructing a Categorization Framework

In this section, we illustrate an example of how a categorization framework is constructed by following the approach above for a real software company. The fully constructed categorization framework is then used in a case study to classify and record process evolution during selected projects in a software company. The full description of this case study is found in chapter 9. Before the construction approach is applied, we present a short summary of the studied software company in the case study.

6.3.4.1 Short Background of the Studied Company

The studied company (referred to as XXX) is a software provider of banking applications. Its complete background and related issues are described in chapter 9. We only summarize some important aspects below for the sake of comprehension. XXX has 329 employees of which 89 are working with software production. The case study is conducted against the System Development Division. XXX is ISO-9001 certified since January 1995 and thus has defined processes for development and maintenance of software systems. These processes (i.e. generic process models) are documented in a Quality System. XXX is organized in several divisions which are responsible to provide adequate services to certain activity areas. A customer application is often realized within a project which is staffed by members from different divisions. At project start, a project mandate is elaborated to coarsely describe the goal and constraints. A project plan (i.e. enactable or enacting process model) which includes full description of activities, their schedules and effort estimates, is then specified by a project manager. The project plan conforms to the globally defined processes which are documented in Project Management and System Development Handbooks.

6.3.4.2 Identifying Questions in Dimension (step 1)

XXX recognizes the high frequency of process evolution and its unpleasant impact on schedule and progress during project execution. It wants to better understand the evolution origin, cause, scope as well as impact on product quality, project delivery, cost and budget. XXX thus wants to have answers to all six questions which are defined in the categorization framework, namely where, why, what, when, how and by-whom. Strictly speaking, the when and by-whom questions are not of relevance according to the stated intention. However, we elaborate them briefly for the sake of completeness. The domain
in this case is a development process in software company. It means that evolution during maintenance, i.e. after initial delivery, is not taken into consideration.

6.3.4.3 Elaborating the Aspects (step 2)

The next step of the construction approach in section 6.3.3 above is determination aspects for each question in the dimension identified in step 1. This is done by elaborating along either the viewpoint or concern axis. Based on the background and the intended use of the categorization framework above, we identify relevant viewpoints and concerns. Relevant roles in this case are developer, project manager and executive. Concerns we need to investigate are technical, managerial and economic. In the following, we determine aspects for each question in the dimension.

Where: the aspects in this question is determined by examining the viewpoint axis in the construction approach. That is, aspects of evolution origins are perceived by various roles in the company.

developer The developer’s major concern is to develop a product for the project within a limited time frame. Developers are mostly affected by evolution which originates from actors internally involving with the project. Changes from actors which influence the project in different extents, has therefore great consequence for the developer.

manager Project manager has often experienced evolution which comes from actors externally involving with the project. The key responsibility of the project manager is to plan, control and coordinate production activities. To do it properly, the project manager should react effectively to changes from project external actors such as customer and eventual subcontractor to steer project in correct direction and to keep project on schedule.

executive Executive is most of the time dealing with evolution which has its origin from factors outside the company to maintain or improve its competitive advantage. That is, company’s external or exogenous factors (e.g. market trend variability, competitor policies, technology availability) and their consequences are of great importance according to the executive’s point of view.

The aspects which are identified in the Where-dimension are thus project internal, project external and external factor.

Why: the aspects in this question are identified by pursuing the concern axis. It means that various aspects of evolution causes are determined to reflect following concerns:

technical It consists of evolution causes which are related to software engineering activities. Such evolution can thus occur during activities in the development life cycle by some unanticipated discrepancies in e.g. functional features, user requirement, or technical competence.

managerial It includes evolution causes which stem from managerial activities. Such activities usually deal with e.g. activity scheduling, estimation, resource allocation and strategic prioritizing rather than directly with software products.

The aspects which are identified in the Why-dimension are thus functional and managerial. Existing literature has identified three reason of change: correction, adaptation, and perfection. All those three reasons of changes are applicable to either technical or managerial concerns above depending on whether the software system or software process is viewed.
6.3. Categorization Framework for Process Evolution

**What:** The *concern* axis is investigated to identify aspects of evolution scope in this question. Interesting concerns are therefore:

*technical* The scope of evolution in this concern mainly affects software products or deliverables which are developed during projects. Such technical artifacts are represented in different formats (report, source code), and sizes (single module, deliverable system).

*managerial* The scope of evolution in this concern influences artifacts which are used to manage project. Some of such artifacts describe standard development procedures and routines for project execution, i.e. generic process models. Others represent established project constraints, e.g. schedule, cost, budget and resource.

*economic* This involves aspects of evolution which affect company’s global profit, earning ability and investment in e.g. purchased software and tools for engineering and managerial activities, i.e. production and project management tools.

The aspects which are identified in the What-dimension are thus *technical artifact, managerial artifacts, constraints* and *software/tool*. In [Leh94], Lehman presents a classification of process evolution, listing five evolution entities: Software Release, Software System under development, Development Process, Process Model, and Application Domain. The first two entities are covered by our technical concern; and the next two by managerial concern. The application domain in our case is confined to development process in a banking software provider (as mentioned in section 6.3.4.2 above).

**When:** The most interesting aspects of this question on evolution time is when a given evolution is detected and then corrected. From the economic *concern*, it is useful to illustrate the fact that the correction cost of an evolution is considerably higher after subsequent development phases. The identified aspects for when-dimension are thus *detection* and *correction*.

**How:** This question deals with the evolution impacts, or with necessary corrective actions to deal with a particular evolution. We choose to elaborate aspects of evolution reactions along the *viewpoint* axis.

*developer/manager* It consists of aspects which concentrate to correct the impact of evolution during project at the very moment it encounters. Typical actions involve manipulation of either activity network scheduling (e.g. rework, allocation extra resources, re-estimation) or activity network topology/layout (e.g. splitting or merging activities). Such actions are therefore characterized as being short-term and urgent (fire extinguisher).

*executive* The executive identifies the problem and evaluates long-term improvement initiatives to prevent the same evolution from occurring again in future projects. Such preventive actions involve long-term investment either in staff training and education or in new technology infusion. Such actions have impacts on and innovate the entire company.

The aspects which are identified in the how-dimension are thus *short-term adjustment* and *organizational innovation*.

**By-whom:** This question seeks to identify the actors who actually correct a given evolution occurrence. The same elaboration along *viewpoint* axis as in where-dimension above is used to identify aspects for this question. However, only developer and manager viewpoints are of interest. It means thus that the identified aspects for by-whom dimension are *project internal* and *project external*. 
Note that different aspects are possibly determined as different viewpoints or concerns have been selected to investigate for the relevant questions in the dimensions.

6.3.4.4 Elaborating the Categories (step 3)

The categories are then determined for each identified aspect above according to the context of the studied company XXX. The common terminologies at XXX are used to describe the categories in the constructed categorization framework. Figure 6.2 depicts a fully detailed categorization framework for process evolution which are specified and elaborated for XXX. The paper-based form which elaborated from this framework, is presented in figure 9.5. The terminologies in categories are regarded as self-explainable, and thus need no further description.

6.4 Project Characterization Framework

In this section, we present a structured framework to characterize software project. As mentioned in the introduction, such ability to explicitly characterize project is crucial to anticipate and effectively handle process evolution. It means that additional effort must be scheduled and incorporated to better react to process evolution in the project plan.

[Boe81] [Fen91] and [Jon91] clearly demonstrate that software process measurement plays a central role in improving productivity and quality of software project. A large number of software metrics have been defined, extensively applied and validated with promising results. However, it is hard to manage and costly to maintain if we start to collect and analyze a large number of measures without a clearly defined goal. Basili has introduced a scientific approach, called Goal-Question-Metric (GQM) [BCR94b] to determine an optimal set of metrics necessary to meet a specified goal. In the following, we combine the GQM and our separation of concern to define a set of measures which satisfy our following goal:

Our goal: to determine a set of measures which have highly consistent in correlation to cost, effort and process evolution in software projects.

Below we present our work on separating relevant concerns of project and its characteristic. We will then elaborate the identified concern of project characteristics to classify project. An example of how our framework is used to classify projects is then described. Finally, we discuss the relation between project characteristics and process evolution.

6.4.1 Separation of Concern

Software project comprises aspects of organization, process and product. The first one represents the context and environment in which the project is carried out. The second one involves the technical and managerial activities which are performed in the project. The last one is the resultant software system of the project. Further, we want to separate project characteristic in three different concerns: profile, estimate, and actual status or outcomes.

project profiles are measures which are usually objective, explicit, certain and partly stable. Their
Figure 6.2: A fully constructed categorization framework for process evolution
characteristic property is the early availability at project start. Due to their inherent characteristics, project profile measures are often used as indicator to predict or anticipate events which likely happen in the future.

Project estimates are measures which are mostly subjective, implicit, uncertain and volatile. However, they are needed in software project to plan, schedule and assess activities. They are elaborated by an estimation process which is both critical and crucial for project management. The estimate accuracy remains unconfirmed until the project is completed.

Project outcomes are the same measures as those in project estimates, but represent project actual performance. They are therefore the most precise data of the three identified concerns. They are used to both assess the estimation capability and to improve the estimates of future projects (e.g. empirical cost projection).

The project profiles are used to predict the behavior of future events, and thus strongly influence the elaboration of project estimates. The project estimates represent and strive to converge to the outcomes of project at completion. In the subsequent section, we determine typical project profile measures which highly influence the project outcomes. The elaborated measures are not meant to be complete and exhaustive. Rather, they can be further decomposed and extended to accommodate particular situation.

6.4.2 Project Profile Measures

In this section, we present an elaboration of the project profile measures based on the three aspects identified above, i.e. organization, process and product. The identified measures are initially known at project start.

Organization the project staff describes participants and their roles in the project. It can include a project manager, a quality manager, analysts, designer, developer, tester, .... Each of participants owns a known competence profile, including education, skill and experience. In addition, the customer profile is also determined at project start. Such customer profile represents e.g. customer technical competence to the problem domain, or our previous experience in working and cooperating with the customer. Similarly, the profile of sub-contractor is also explicitly known, and expresses e.g. its capability to deliver, and its expected product quality.

Process the type/purpose measure classifies project after various categories, e.g. development, upgrading, maintenance. Different project types represent different degrees of skill, technical competence and knowledge about a problem domain at hand. The concern represents the overall strategic goal which is defined at project start. The risk measure specifies issues and their extents to which the project likely fail or succeed. The technology measure indicates the project participants' competence of and knowledge about the adopted technology.

Product the application domain specifies usage area of the software product, while type represents the language and platform the software product is developed upon.

Figure 6.3 depicts the project profile measures which are described above. The number of identified project profile measures is not exhaustive. Several characteristics can also be incorporated according to specific needs, e.g.
6.4. Project Characterization Framework

Figure 6.3: Hierarchy of project profile measures

- Organization’s capability maturity level based on SEI’s Capability Maturity Model [PCCW93a].
- Enterprise size (e.g. big, middle-size or small).
- Newness of the product: release versus new software; well-established, familiar versus new application domain; etc.

The hierarchy of project profile in figure 6.3 has been used to determine relevant measures to characterize projects in our case study in chapter 9. The paper-based form to characterize projects in this case study is found in figure 9.4.

6.4.3 An Example of Project Characterization

In this section, we illustrate an example in which we characterize different project types according to the project profile measures depicted in figure 6.3. The example illustrates that the hierarchy of project profile measures is applicable to characterize following four typical project types: (see also table 6.1).

1. A classical project is constrained by limited budget and duration. The project usually has a specific customer who initially has a set of clear requirements. The frequency of staff-turnover is not high in such project, and the staff competence is sufficient to solve problems in a particular domain. There are typically development and maintenance projects whose main concern is to deliver product on schedule, on budget, of high quality, and satisfy customer. Products are often application software or tailored software.

2. A least-time project has a major concern to put the product to the market as soon as possible regardless of product quality. By occupying the market with their products, they can gain increasing market share, and thus growing earning. The staff is often well qualified and talent to work efficiently and even under time pressure.
3. A critical project is, apart from limited budget and schedule as a classical project, required to produce high quality software. Their products are used in application domains in which aspects such as time, reliability and error are critical, and people life may be sacrificed if those quality requirements do not meet. Such projects are often funded by government to produce software for e.g. military, space research, nuclear power plant station, ....

4. A research project has no intention to develop a commercial product, rather to test or validate new ideas or methods in science, computer science, software technology, .... The frequency of staff turn-over is high due to long duration of such projects and the increasing demand of research competence in the software industry. The staff consists of motivated and skilled idealist. The project schedule and budget is not critical, and carries out with very low risk.

<table>
<thead>
<tr>
<th>aspect</th>
<th>projects</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 classical time/budget constrained projects</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>organization</td>
<td>customer</td>
</tr>
<tr>
<td></td>
<td>staff</td>
</tr>
<tr>
<td>purpose/type</td>
<td>develop/maintain application software in a particular domain</td>
</tr>
<tr>
<td>process</td>
<td>run the project within budget, deliver software in time, user satisfaction</td>
</tr>
<tr>
<td>concern</td>
<td></td>
</tr>
<tr>
<td>risk</td>
<td>schedule delayed, budget overrun, user’s complaints</td>
</tr>
<tr>
<td>technology</td>
<td>either well–known or partly known</td>
</tr>
<tr>
<td>product domain</td>
<td>application software, tailored software</td>
</tr>
<tr>
<td></td>
<td>commerce, industry, medicine, communication,</td>
</tr>
</tbody>
</table>

Table 6.1: An Example of Project Characterization

6.5 Relation between Process Evolution and Project Profile

Software production is best characterized as an experimental and evolutionary discipline. Software projects therefore differ greatly in their initial assumptions, environmental constraints, premises, and context. Not two projects are alike. Captured process evolution occurrences are thus also different from one project to another. However, DeMarco has stated in [DeM82] that projects may differ qualitatively, but many or most component pieces of the project will not. Further, Jones has in [Jon96] observed thousands of projects within hundreds of organizations and drawn a following significant conclusion. “While there are myriad ways to fail when building large software systems, successful projects surprisingly all tend to follow similar patterns of development practices regardless of what application domains or country develops them”. The two statements above imply two things:
1. It is possible to determine a minimal set of profile measures which best distinguish projects from each other. That is, the hierarchy of profile measures in figure 6.3 should be further extended or restricted to adapt to the specific project type in the organization.

2. It is likely to find consistent relations between the identified profile measures and process evolution patterns. Such relations can not be established over the night, but rather by continuously collecting and analyzing process evolution during several projects.

Below, we outline some requirements for project profile measures.

- Project profile measures must be available at the project start. They should be complete, certain and preferably not changing very much during the project.

- Project profile measures should be able to distinguish major similarities and differences between projects. It is therefore important to determine a minimal set of profile measures to meet such a goal in a cost-effective way.

- Project profile measures must illustrate highly consistent in correlation to cost, effort and evolution. That is, there exist significant evidences for convergence between them on one hand and project outcome measures and process evolution patterns on the other hand. It is not possible to determine such profile measures at start, but we gradually gain insight and knowledge about that over time.

The identified project profile measures may have different extents of impacts on the process evolution. E.g., the customer profile has strong influence to the frequency of a certain process evolution because projects in this company require great deal of customer involvement. To reflect such different degree of impacts on process evolution, profile metrics are therefore associated to a weight. Such a weight, which is corporate specific, also indicates the dominance of a given profile metric when the degree of similarity between projects are determined. In chapter 7 later, we present a detailed algorithm to select best matching completed project based on those weights of profile measures.

We are not only interested in knowing the frequency of process evolution in a given project type, but also its cost in term of money or additional working hours. Each recorded process evolution occurrence is therefore associated with a cost measure. This cost measure can have both estimated and actual values.

6.6 Chapter Summary

In this chapter we have presented a framework to categorize process evolution. This categorization is significant to better understand the property as well as the potential impact of process evolution occurrences. The basic structure of the framework consists of three main concepts: Dimension, Aspect and Category. The categorization framework is explicit, comprehensive, evolved, abstract and in continuous use. We have also described an approach to construct a categorization framework according a specific context. An example of such construction has also been illustrated. The fully constructed categorization framework is then used in the case study in chapter 9. We believe that it is not realistic to only categorize process evolution without linking them to a particular project context. Such a
belief has motivated us to define a project characterization framework which is constituted by three relevant aspects (i.e. organization, process and product) of a project, and three concerns (profile, estimate and outcome) of a project characteristic. Further, we have elaborated in details the concern of project profile, and identified a set of profile measures which are organized as a hierarchy. This hierarchy of project profile measures has also been exploited to determine relevant measures to characterize projects in the case study. The identified profile measures have then been used to classify four typical software projects. Finally, we present a discussion on the relation between project and process evolution and the different impacts of project profile measures on project outcomes and process evolution. To better represent such varied extent of impacts, we associate each project profile measure with a weight. Such weights are used to determine similarities between projects (see more about this in the discussion of approaches in chapter 7).
Chapter 7

Approaches to Manage Process Evolution

7.1 Introduction

In this chapter, we present approaches in EPOS to manage software process evolution which affects all three process model variants. We start with a discussion of basic decisions behind our approaches in section 7.2, followed by a brief description of the overall meta-process model which inspires our work in section 7.3. Section 7.4 describes an approach to manage evolution of generic process model. In section 7.5, we discuss different approaches to handle with process evolution which implies modification of enactable and enacting process models. A brief discussion on how to evolve real-world process is outlined in section 7.6. Section 7.7 presents an approach to establish and reuse empirical evolution experience in planning a new project. The empirical evolution planning approach relies on the categorization framework which is presented in chapter 6. We then outline a set of requirements for tool support in section 7.8. An evaluation of the proposed approaches is discussed in section 7.9.

7.2 Basic Decisions

As mentioned in previous chapters, there are three variants of process models which represent the software processes in the real world. They are generic, enactable and enacting process models. By explicitly describing the real-world processes, their performances are effectively guided and assisted by a set of process tools. The three process model variants, process modeling language (PML) and process tools constitute process support for PSEE. The generic process model (or process model template) is most abstract, superior, and represents the software processes at the company level. It usually describes the company’s overall policy, strategic goals, standard procedures, routines to develop or maintain software products (i.e. production processes) as well as to evolve software processes (i.e. meta-processes). The enactable process model is then specialized by the generic one to incorporate specific project constraints and premises. It means that a computerized representation of a particular project plan is an enactable process model. The project plan describes i) project participants together with their roles and responsibilities; ii) activities together with their duration, ordering, decomposition and inter-dependencies; iii) planned resources scheduled to various activities. When the project plan is carried out and continuously evolved to adapt to changes during project, its representation is referred to as an enacting process model. The process tools in a PSEE assist the software company to define generic process models, to establish project-level enactable process models, to execute and evolve enacting process models. In that way, the performance of both production and meta-processes
are conforming to defined models whose ability to deliver high quality products on time and on budget has been explicitly demonstrated. Bearing that in mind, it is important to keep process models and real-world processes synchronized at all times. That is, process evolution occurring during projects should be effectively incorporated and reflected in all three process model variants.

Until recently, most research effort in process community has focused to address evolution of generic process models. Many approaches and architectural solutions have been defined and implemented in several PSEEs to adapt generic process models according to changes during projects. Such evolution is often a result of e.g. an infusion of new technology to improve the existing production process, and should thus be carefully planned. The evolution is then propagated to the two corresponding process model variants. Such propagation of changes in generic process models can be carried out in either eager or lazy manner. Eager propagation makes new changes visible to “living” enactable and enacting process models, while lazy propagation only incorporates new changes to newly created ones. We therefore call such changes top-down process evolution.

However, we claim that process evolution affecting generic process models occurs more seldom than those unanticipated changing enactable and enacting process models. Our claim will be validated by a case study examining real projects in a software company (see chapter 9). The case study results clearly demonstrate a need for adequate support to deal with unpredictable changes to project plan before and during execution. The characteristic feature of such changes is that they are not necessarily packaged in the generic process models. They rather reflect some project-specific urgent situations, e.g. scheduling more time/resources for an activity; or adding/removing activities to deal with a specific incident. Such changes are effective during the project life time, and are possibly packaged up to corresponding generic process models at completion if they are considered as important for process performance in the future. We therefore refer to such changes as bottom-up process evolution. One may wonder whether project management tools are designed to deal with such changes (cf. discussion in chapter 2). It has also been shown that existing project management tools do not provide appropriate support because of high frequency of evolution and re-planning in the software process. We therefore need to integrate project management supports with process management capability of PSEE to better manage the bottom-up process evolution.

### 7.3 Overall Meta-process

The elaboration of this generic meta-process model (also referred to as process life cycle) which is published in [CJ93, CFF94], is inspired by the Quality Improvement Paradigm (QIP) [Bac93], [BCR94b], Plan-Do-Check-Act paradigm in [], and the traditional software life cycles (e.g. waterfall).

We summarize important aspects of the generic meta-process model. It includes five main meta-activities to both create and modify new and existing process models respectively.

**Technology provision** This meta-activity is in charge of defining an appropriate process modeling language to describe basic process models, and of providing a set of process tools.

**Process requirement analysis** This meta-activity takes into account the existing process and their corresponding models to provide (new) requirements to the design meta-activity.

**Process design** This meta-activity provides the general and detailed architecture of the process. An
important input to this meta-activity is the description of the technology that has to be integrated and used within the new processes.

**Process Implementation** This meta-activity is in charge of implementing the design specification. It involves changes to both process model variants and real-world processes.

**Process assessment** This meta-activity provides quantitative and qualitative information describing the performance of the whole process. Such information is used by the process requirement analysis meta-activity.

### 7.4 Evolution of Generic Process Model in EPOS

In this section, we propose an approach in EPOS to manage generic process model evolution. This approach is a specialization of the overall meta-process model (in section 7.3). It is included in this chapter simply for the sake of continuity for later sections. Our specific approach for managing generic process model evolution together with associated tool support in EPOS [NC94b] is depicted in figure 7.1. Below, we present a brief description of the meta-activities.

![Diagram of meta-activities to evolve generic process model in EPOS](image)

**Figure 7.1: Meta-activities to evolve generic process model in EPOS**

**Monitor:** This meta-activity continuously collects feedback, i.e. relevant measures from the real-world process performance, to prepare for the next meta-activity which detects poor and undesirable behavior. That is, the monitoring meta-activity is in charge of collecting relevant measures from
the production process such as effort spent for each activity; number of error found from a particular document type. In our current implementation, this monitoring feature is done by an embedded monitor-task which simply receives the amount of effort spent (in term of hour) by each task in the task network. Currently, each task has to explicitly report its consuming effort to this monitor-task at completion.

Propose: This meta-activity analyzes the collected measures from the monitor-task to identify deficiencies based on some predefined criteria. E.g. if the actual effort of performing a given task exceeds a defined limit, its associated process model is automatically regarded to be inefficient. This meta-activity has as input a knowledge about existing technology and currently best practices in a particular domain. The objective of this human-intensive meta-activity is to propose a process change request (PCR). Such requests comprise suggestions which can vary from adoption a new tool, to infusion a new development technology. Therefore, this meta-activity requires a great deal of creativity and technical insights.

Analyze impact: The informal PCR is then analyzed to identify potential impacts. The relevant impacts which we are basically concerned with, are e.g. the cost versus the improvement effect of, and the risk to implement the PCR. The answers can be retrieved by conducting a series of experiments which will take a lot of time. Instead of aiming at such goal, we are simply interested to investigate the extent to which existing process model variants are affected by the new change. That is, we want to restrict the impact of new changes within certain extent. The change impact is determined and evaluated by the Schema Manager (see more in section 4.5.6.1). A formal PCR is sent to next meta-activity if the change is approved.

Implement: The existing generic process models are augmented with new suggestions of changes in the formal PCR. This is done by using the Schema Manager which provides support for defining or modifying new and existing process models respectively. The Schema Manager also analyzes the new model fragments to detect both syntactic flaws and semantic inconsistencies. We adopt late and dynamic type binding in EPOS. Therefore, new changes in generic process models (i.e. types) are immediately visible for enactable and enacting process models (eager change propagation). It means that it is possible to propagate to the active enacting process model by first suspending it, and then changing its corresponding generic process models. By doing such propagation, the task network (i.e. enacting process model) is not consistent anymore and thus requires re-planning. An incremental re-planning algorithm [LC93] is carried out by the Planner.

Plan: This meta-activity is in charge of instantiating an enactable process model from a generic one. That is, the enactable process model is constructed or derived from the generic one to satisfy a specific goal. This meta-activity is carried out by invoking the Planner which automatically and incrementally constructs the task network ready to be interpreted and executed (see Planner in section 4.5.5.2).

Enact: This meta-activity is in charge of interpreting and executing the enacting process model. This meta-activity is performed by the Process Engine (section 4.5.5.1) to guide and support the real-world process performance.

Similar to other PSEEs, we have put a great deal of effort to address evolution on the generic process model level. Of course, the approach also changes the enactable and enacting process models by propagating new changes done in the generic process models. Such approach therefore deals with top-down
process model evolution. As mentioned above, such modification of all three process models variants does not occur so often. Instead, there are many cases in which only the enactable and enacting process models are needed to be changed. The next section presents an approach to address such evolution.

7.5 Evolution of Enactable and Enacting Process Model in EPOS

In this section, we present an approach to evolve enactable and enacting process models in EPOS. This approach is in contrast with the one in section 7.4 performed in a bottom-up manner. That is, the basic evolving entities are in this approach enactable and enacting process models. The new changes remain in the affected entities during their life times. Such changes are also referred to as local customization in the Conradi and et al's generic meta-process model (summarized in section 7.4). In addition, the approach provides flexible support for propagating new changes to enacting variants upward to generic ones. Such upward propagation of changes occurs when the new changes to enactable/enacting process models appear to be more efficient than the corresponding ones.

We identify two approaches to evolve enactable and enacting process models in EPOS. They manipulate the EPOS's task network layout either before or during enactment; and task network scheduling. In the following sections, we will discuss them in depth.

7.5.1 Manipulation of Task Network Layout

As mentioned in chapter 4, task network formalism is adopted in EPOS to represent enactable and enacting process model variants. The enactable one is constructed by the Planner, while the Process Engine interprets and execute the enacting one. The Planner provides automatic planning of the task network based on existing types (i.e. generic process models), an initial goal of a decomposable task, and a product structure (i.e. a set of product instances coupled by defined relations). The goal includes a set of product states that has to be achieved by executing the decomposable task. However, fully automatic planning of the task network has shown to be unrealistic in practice. Therefore, we need to manually manipulate the constructed task network both before and during enactment without affecting the generic process models. Such functionalities have not been supported by the current EPOS. We identify two basic operations to manipulate the task network layout: add a new task to, and remove an existing task from the task network. Tasks can be divided or merged by combining various sequences of those two basic operations. In the following, we describe them in details.

7.5.1.1 Task Insertion Operation

Figure 7.2 depicts a situation in which a new task $T_4$ is inserted to the existing task network. Before insertion the task network includes a parent task $T$ and its subtasks $T_1$, $T_2$, $T_3$. We refer to the sequence of those subtasks as a subplan of the parent task $T$.

A set of constraints we have to take into consideration when a new task is added to the existing task network. They are:

- The type definition of the new task must exist.
The new task can only be added at the end or middle of the subplan of the parent task. Addition of a task at the beginning of a subplan is forbidden because it violates the initial premise of the parent task. In other word, the new task may introduce new products which are not available for the subplan. On the other hand, addition a new task at the end of the subplan implies that more (superfluous) products are generated than original goal of the parent task. We perceive that such an operation is not harmful and thus allowed.

At least one input of the new task must match with one of the outputs of the preceding task. Similarly, at least one output of the new task must correspond to one of the inputs of the subsequent task. The second requirement is not necessary if the new task is inserted at the end of the subplan.

If the new task is added to the task network during enactment (i.e. enacting process model), the insertion is restricted to only the part of subplan which is not executed yet, i.e. the part of the subplan subsequent to currently executing task. The task insertion example in figure 7.2 is only possible when T1 is currently executed and T2, T3 are waiting. From an implementation point of view, it is however feasible to add new task everywhere in the subplan. But it then implies a potential risk for that the new task is never executed. This constraint is define to assure that new task will be executed at least once in the future.

When adding a new task to the existing task network, a set of type-level attributes in the process models are affected. A task insertion implies changes to following type-level attributes:

- **DECOMPOSITION** represents the vertical task network breakdown and contains a pool of task type candidates of decomposed subplan of the high-level parent task. Adding a new task to the subplan implies that the task pool must be extended with the new task type.

- **PRE_STATIC** condition describes the required states of inputs to a given task. i.e. entry criteria. This condition must be adjusted to reflect the new states of task inputs.
• **POST_STATIC** specifies the required states of outputs to a given task, i.e. exit criteria. This condition is violated because it has to produce outputs requesting new states.

• **FORMALS** defines the horizontal chaining of the task network, specifying expected types for inputs and outputs to a given task. Tasks which consume new products generated by the new task, must have its FORMALS modified.

• **PRE_DYNAMIC** condition is evaluated by the Process Engine to determine whether the task is ready to be executed or not. The PRE_DYNAMIC condition has two parts. The non-temporal is not affected by the manipulating operations of task network layout. The temporal part of PRE_DYNAMIC is specified by a boolean expression of required states of input in PRE_STATIC.

• **CODE** defines the sequence of actions performed when the task is executed, i.e. when its PRE_DYNAMIC is evaluated to true. The semantic change to CODE must be performed by human to adapt to new situation.

The first four type-level attributes are used by Planner to construct a subplan, while the last two are used by Process Engine to interpret and execute the task network. The type-level attributes in EPOS facilitate dynamic and late binding when type definitions are changed (i.e. top-down process evolution described above). Our process tools (e.g. Planner, Process Engine) therefore operate on those type-level attributes to incorporate changes on-the-fly (i.e. eager propagation strategy). As mentioned above, we want to manipulate the task network without changing the generic process models. To remedy this problem two adjustments must be done:

1. Those affected type-level attributes above must possibly be redefined at instance-level. That is, each task has instance-level attributes for DECOMPOSITION, PRE_STATIC, POST_STATIC, FORMALS, PRE_DYNAMIC and CODE. They are initially empty as default.

2. The Planner and Process Engine must be modified to use those instance-level attributes when they are non-empty. Otherwise, they can use the corresponding type-level attributes as usual.

To easily illustrate the approach, we introduce a graphical notation to clarify EPOS concepts. An **AND-port** connects inputs or outputs to a task. The PRE_STATIC condition is represented by attributes of the AND-ports at the input side of the task. Similarly, the attributes of the AND-ports at the output side of the task specify the POST_STATIC condition (see figure 7.3). The temporal part of PRE_DYNAMIC condition can then be represented by assigning inputs to different AND-ports to specify a boolean expression. That is, the AND-port is true if its inputs achieve the required states simultaneously. The task is activated by at least one true AND-port. In the same manner, the alternative task outcomes can be expressed by assigning outputs to various AND-ports. It means that the task is completed when at least one AND-port at the output side is true.

The task insertion operation can be carried out in following steps:

1. The location in the subplan where the new task is inserted, must first be selected. The location must be evaluated against the constraints above to determine if the operation is allowed. Otherwise, the operation is aborted.

2. The new task type must be selected among existing ones. This thus assures the type existence constraint mentioned above.
3. The type-level attribute FORMALS of the new task type is examined to see whether its input and output types partly correspond to the output types of preceding and input types of subsequent tasks respectively. If this is not a case, the insertion is denied.

4. An instance of the new task type is created. Its matching inputs and outputs are automatically linked to those of preceding and subsequent tasks. If its non-matching inputs and outputs are among existing products, they are then linked together. Otherwise, new products are created. The newly created outputs should be used by other tasks, and are therefore manually directed to one of the sibling tasks in the subplan.

5. When the graphical insertion of task is completed, the task network is still semantically inconsistent. The user is notified about the possible inconsistencies, and encouraged to adjust them. Possible inconsistencies which need to be revised, are:

   - PRE_ and POST_STATIC conditions of either the preceding or the subsequent task. The extent of adjustment depends on the definition of PRE_ and POST_STATIC conditions in the new task type.
   - PRE_DYNAMIC condition, FORMALS and CODE of the task which receives and consumes the new task’s output.
   - DECOMPOSITION of the parent task of the subplan in which the new task is inserted.

6. The insertion operation is not completed before all above steps have been performed. That is, the task network must resume to a consistent state before continuing execution.

The task insertion operation is illustrated by a scenario in figure 7.4. A review task is added between an edit and a compile task. Such insertion probably aims to reduce the number of errors at compilation.

The review task type is already defined among existing types. Its FORMALS type-level attribute corresponds with those defined in edit and compile task types. The matched source code object is reconciled with output and input of edit and compile respectively. The design document input
7.5. Evolution of Enactable and Enacting Process Model in EPOS

Figure 7.4: Example of adding Review to the task network

To review task is an existing product and then automatically linked to the new task. On the other hand, the review_feedback output of review task is a new product and is thus created. The user is notified about this new output and asked to determine which tasks in the subplan is going to receive the review_feedback. In this case, the non-matched review_feedback is used by the edit task. Such explicit designation of non-matched products is context and semantic dependent, and is thus done by human.

It then implies a conflict in PRE_STATIC of compile. The PRE_STATIC attribute in compile's AND-port must be revised from [edited] to [reviewed]. The PRE_DYNAMIC of the edit task should be extended to express the following condition: edit is executed when either the design document and the source code both have created state; or the feedback from compilation has initiated state; or the review_feedback has initiated state. Such extension is done by creating a new AND-port with review_feedback to edit task. The CODE and FORMAL of edit task must also be modified to take into account the existence of the new input. Finally, the parent simple task has also updated its DECOMPOSITION to reflect the new subplan. The CODE modification is manually performed to decide what to do with the new input. Except from the modification of CODE, other modifications can be partly automatic or supported by tools.

Note that all the changes described above are done to the instance-level attributes, and thus do not affect the type information (i.e. generic process models). Those instance-level attributes are used by Planner and especially Process Engine to execute the task network further. New instance-level attributes are kept during the life time of the task network (i.e. enactable or enacting process model). When the task network is about to be destroyed (i.e. the project is completed), there is possibility to package those
instance-level attributes to the corresponding generic process models. We are then back to the top- down process model evolution approach which is described in section 7.4 above.

7.5.1.2 Task Removal Operation

Figure 7.5 illustrates a situation in which a task $T_2$ is removed from the existing task network. The task network after such operation is expected to sustain consistency as well as correct execution flow. That is, the new task network is able to continue to achieve its defined goal.

![Diagram of task network](image)

Figure 7.5: Removing task T2 from the task network

To meet the expectation above, we specify below a set of constraints for task removal operation.

- It is only possible to remove tasks which are resided in middle of the subplan. That is, the starting and final tasks of the subplan is not removable. This constraint assures that the sub-plan's original premises and goals remain unchanged. This constraint is defined to restrict the semantic violation to the process models. From the implementation point of view, it is definitely feasible to remove any tasks in the task network.

- The preceding and subsequent tasks of the deleted task must be able to link to each other by at least a common product. This constraint makes sure that the execution flow of the new task network is retained after the task removal operation.

The discussion about the affected type-level attributes and necessary adjustments in task insertion operation is still valid for the task removal operation. The approach to remove a task from the task network is then described in following steps:

1. The deleted task is selected. Its location is evaluated against the constraint above to determine whether the removal operation is permitted. If not, the operation is canceled.

2. The FORMALS of preceding and subsequent task types are examined to see whether a connection between them can be established. In concrete, at least an output type of the preceding task must correspond to an input type of the subsequent one. If no match is found, the operation is abandoned.
3. The matching outputs and inputs are automatically connected together, and the selected task is removed. The non-matching inputs and outputs of the deleted task are thus superfluous. The superfluous inputs can be safely removed because the deleted task only reads the input, and its absence does not affect other sibling tasks. In the contrary, the superfluous outputs must be taken care of. All tasks which consume the superfluous output and thus are affected, are identified.

4. The tasks are now connected, but the task network is still semantically incorrect. The user is notified about the affected tasks and implied inconsistencies. Possible inconsistencies which need to be corrected, are:

- PRE_STATIC and POST_STATIC of the preceding and subsequent tasks between the deleted task.
- PRE_DYNAMIC condition, PRE_STATIC condition, FORMALS and CODE of the affected tasks which consume the deleted task’s superfluous outputs.
- DECOMPOSITION of the parent task of the subplan in which the the task is removed.

5. The removal operation is not completed before all above steps have been performed. That is, the task network must resume to a consistent state.

The task removal operation is illustrated by a scenario in figure 7.6. The task review is decided to be removed from the task network. Such task removal is probably due to lack of resources to review the source code before compilation.

![Diagram of task network removal](image)

Figure 7.6: Example of removing Review from the task network

The position of the review task implies that the removal operation is allowed. The FORMALS of edit and compile task types are examined to determine whether they can be connected by at least
a common product. It appears that the matched source code can link them together. The superfluous input (i.e. design document) can be safely removed without implying any inconsistencies. On the contrary, the superfluous output (i.e. review_feedback) is used by the edit task.

The user is notified about the inconsistencies in the edit task whose FORMALS, CODE, PRE_DYNAMIC, and PRE_STATIC must be modified to deal with the deleted review_feedback (e.g. removing its associated AND-port). It is only necessary to revise the PRE_STATIC of the compile task (i.e. from [reviewed] to [edited]) to match with the POST_STATIC of the edit task.

Again, it is worthwhile to emphasize that all changes to task instance-level attributes are intact during the task network life time without modifying the generic process models. Of course, those new changes can be incorporated to corresponding generic process models at explicit demand.

### 7.5.2 Manipulation of Task network Scheduling

We have seen above how to manipulate the task network layout to reflect the evolutionary nature of a software project. However, it is not the end of the story. Sometimes, it is required to modify the schedule of the task network to adapt to various change incidents and unanticipated circumstances. They can vary from a need to revise initial effort estimates, to a need to re-execute or to postpone a start time of a particular task.

In EPOS, the Planner constructs a task network which only determines the vertical work breakdown and horizontal execution ordering of activities without associating any time aspects to them. It means that a task can keep executing within an unlimited time frame without being interrupted or its owner being warned. This scenario is not realistic for a software project in the real-world. Both before and during enactment, we therefore do have a need to continuously revise such time attributes to reflect such a changing project environment. Following issues must thus be addressed in the current process models and process tools in EPOS.

- New instance-level time-related attributes such as start time (S), elapse time (E) and duration (D) must be defined and associated to a task. Such instance-level attributes are expressed in appropriate time units (e.g. minute, hour or day).
- New instance-level relation is defined to connect between a task and a person. In current EPOS implementation, we have currently only a type-level relation to restrict the performance of a particular task type to a specific human role.
- Two new task states are introduced. They are suspended and canceled which indicate a temporary or permanent lack of resources to perform a particular task. That is, the suspended task is occasionally resumed, while the canceled task is discarded.

Following operations are considered necessary to support dynamic manipulation of the task network schedule.

### 7.5.2.1 Revision of Task Schedule

The Planner constructs a task network with a coarse-grained schedule based on existing project constraints (i.e. total duration). The automatic scheduling facility of the Planner relies on process informa-
tion and actual performance of completed projects. The Planner thus provides an initial suggestion of schedule for the new project. The (S,E,D) time attributes which are associated to tasks in the project, can then be revised manually before the task network is enacted. I.e., the initial schedule is adjusted to reflect human (e.g. project manager) insight and/or to incorporate project specific constraints. Those estimates are likely over- or under-estimated due to limited knowledge. Therefore, the triple time attributes (S,E,D) should necessarily be modified during enactment when more knowledge is gained. In that case, it is important to keep the task network in consistent state after a task schedule is revised, i.e. the triple (S,E,D) of other tasks must be adjusted accordingly. After such schedule revision operation, a new triple (S,E,D) is created to keep track of the execution history of the task. The last (S,E,D) will therefore represent the actual duration of the associated task. Below, we distinguish three cases and related constraints when the task schedule is revised.

1. If the \textit{elapse} time is equal to 0, the associated task is not yet started (i.e. forthcoming task). It is only possible to change the \textit{start} time and \textit{duration} of such forthcoming tasks. Changing the \textit{start} time implies either to postpone or advance the task execution. The effort estimate of the task is revised if the \textit{duration} is modified.

2. If the \textit{elapse} time is different from 0 and is smaller than \textit{duration}, the task is executing (i.e. ongoing task). In this case, it is only possible to change the \textit{duration} value to revise the effort estimate. The new value of \textit{duration} must be higher than the one of \textit{elapse} time. It means that additional effort is assigned to the particular task.

3. If the \textit{elapse} time is equal to \textit{duration}, the task is completed. All three attribute values can be changed to make the completed task to be re-executed (i.e. rework). The new \textit{duration} value is the sum of the existing one and the estimated duration needed for the rework. The \textit{elapse} time is set to 0, and the \textit{start} time is assigned an appropriate value. By examining the execution history of a particular task, the number of rework iterations can be determined by counting the number of triple (S,E,D) with \textit{duration} value equal to 0.

\section*{7.5.2.2 Revision of Task Resource}

A particular task is initially decided to be carried out by a person. Such assignment information is determined and possibly evolved before and during task network enactment. Such resource revision operation is constrained by the resource availability at a specific moment. A \textit{resource pool} specifies e.g. either occupied or free human resource in the project staff at a particular moment. Such issue is beyond our research scope and will not be treated here. On the other hand, it is necessary to re-allocate project staff from one task to another in case some unanticipated events occur, e.g. sickness, resignation, .... In concrete, we are interested in performing two following operations for the ongoing and forthcoming tasks in the task network.

1. Changing the performer of a task. The task must be either executing or planned to be carried out in the future. The existing performer is replaced by one of the available participants in the project. The task keeps executing after this operation.

2. Changing the execution state of a task. The performer of an ongoing task is somehow not available and the task must be either suspended or canceled. The tasks which are dependent in the suspended task, must adjust their execution states accordingly. Suspending a task can be
due to lack of resource or information/data to proceed its performance. In such a case, the triple (S,E,D) must be frozen until the task is explicitly resumed.

7.6 Evolution of Real-world Process

We have in previous sections discussed approaches to evolve the process support, i.e. the three process model variants. None has been said about how the real-world processes are changed. We have also distinguished two kinds of process model evolution: 1) top-down evolution initially changes the generic process models, and propagates to new changes to enactable and enacting ones by either eager or lazy strategies. 2) bottom-up evolution directly changes to the enactable or enacting process models, and possibly reflects in the corresponding generic ones sometime later. Both evolution types stem from the recognition of some undesirable or deficient features of existing real-world processes. Top-down evolution can be regarded as a major innovation (i.e. process improvement), while bottom-up evolution is characterized as continuous adaptation and eventual learning of change.

In the case of top-down process model evolution, we have to think about the transition process from the current practices to the new ones. That is, the new changes in the process models, e.g. an infusion of new development methodology or adoption of new tools, must be carefully planned and performed in the real-world. The transition process involves technical aspects such as the introduction of new tools or methods. It can also include non-technical aspects such as user training and motivation. The organization culture, social relationship and the way of thinking and doing thing must also be changed. Such transition process is complex and likely out of the scope of a PSEE. To summarize, the top-down process evolution starts with a model evolution and then a transition process. It involves both large and small scale process improvement.

In the case of bottom-up process model evolution, the process models are adapted to the changes occurring during project execution. That is, changes are taking place before they are reflected in the models. We are not necessarily concerned about the transition process as described above. However, it should be emphasized that changes which are incorporated during projects, do not have the same innovative characteristic as those in the top-down process model evolution. It is due to the limited resources and time in project to implement such major improvement. If new changes which are done to enactable and enacting process models demonstrate to be more efficient than the existing generic ones, the new changes will be incorporated. That is, bottom-up process evolution involves an initial experiment before evolution of internal process model. Thus, the top-down and bottom-up process evolution is two sides of the same coin.

In common, the evolution of the process models and the changes in the real-world process must be consistently synchronized. This is a major prerequisite for that a PSEE still provides adequate support for guiding and supporting process performance. However, it is easier to evolve the models than the real-world in which a great deal of social factors (e.g. human being, mentality, social relation, organization culture, etc.) involve. E.g. infusion a new technology implies total change to the usual way of working and thinking to affected personnel. Evolution of real-world processes turns out to be a difficult, but important issue to manage. It requires full commitment from the top management level to the grass-roots within the organization. To do that, people must be convinced by clearly demonstrating the rationale or the tangible benefits of changes. People is not expected to change over night. Thus, it requires up-front investment, long-term patience and continuous commitment.
We have experience the difficulty in introducing new technology to the real-world processes in our case study with a software organization (more detailed in section 9.5 and chapter 9). Our initial intention was to convince the organization about the benefit from establishing an experience database and its planning support. We were so optimistic that we defined an unrealistic goal for such a short period of time. Although we have managed to introduce the concepts, rationale and tool support for our ideas, the experience database has not been actively used by project personnel. It is probably due to lack of both full commitment from upper management and few tangible benefits after such short period of time. In addition, it clearly demonstrates the resistance and poor susceptibility of personnel to new changes in the real-world processes.

### 7.7 Empirical Evolution Planning Approach

We have presented in previous sections approaches to manage evolution of three different process model variants. The approaches are also distinguished in top-down or bottom-up approaches. The top-down evolution approach detects undesirable properties in existing processes and then improves them by e.g. infusing new tool and technology. Such poor process features are discovered by conducting process assessments based on methods, e.g. CMM, SPICE. Such methods rely on the current best practices in a particular application domain, and specify a set of improvement guidelines which assume to be globally applicable for a wide spectrum of software organizations. On the other hand, the bottom-up approach addresses local changes which eventually occur both before and during project execution. That is, it starts to deal with a problem when it first is identified. Both manner of approaches aim at improving the current process by either comparing it against an universal, widely accepted best practices, or adapting it to consecutive changes.

In this section, we present an approach which elaborates the two viewpoints and helps to improve predictability based on own experiences of evolution from completed project within an organization. The existing process is improved by assessing and learning from own past experience rather than from global methods as in top-down approach. Changes are anticipated rather than just dynamically handled as in bottom-up approach. The approach relies on empirical data on evolution and then assists in planning new projects. That is the reason why we call empirical evolution planning approach. The approach adopts our evolution categorization framework, and is a combination of traditional process planning technology [COWL91, LC93, CHL94b] and process measurement and experience reuse [BR91, BCR94c]. The approach for empirical evolution planning includes three major steps which are also discussed in following sections.

1. **Establish baselines** This step consists of activities collect evolution occurrences from completed projects. Collected data is structured and organized in relevant baselines for future projection.

2. **Support initial planning** The established baselines are used to initially plan a new project. The effort estimates are elaborated based on experience from similar completed projects. It means that evolution which likely occur, is anticipated and corresponding reaction effort is incorporated in the project plan.

3. **Support iterative planning** The initial project plan is however unable to predict all evolution occurrences during the new project. The project plan is thus necessary to be continuously revised to adapt to unanticipated evolution.
7.7.1 Evolution Baseline Establishment

This step includes activities which continuously provide and calibrate evolution baselines from completed projects. They can be considered as a never-ending organizational learning process. Such learning process always play a crucial role, especially the initial phase in which neither baselines nor experiences on evolution exists. However, this process must be repeated to update and enrich existing organizational knowledge as projects are carried out and completed. The evolution baselines are established by performing following steps.

1. The categorization framework which is presented in chapter 6, must be constructed to reflect the particular software organization according to the algorithm presented in section 6.3.3.

2. A set of representative project profile measures are identified to best characterize development projects in the organization. Each project profile measure is assigned a corporate-specific weight (see section 6.5 for detail). These weights which are useful to determine similarities between projects, can be revised as new insights are gained.

3. The categorization framework and project characteristics are then incorporated into the development process. That is, they are ready to be used to collect evolution and its impact during new project preferably without any major additional effort. Appropriate tools or paper-based forms for collecting necessary data must be established and properly installed.

4. The categorization framework and the project characterization framework are then used to collect and classify evolution occurrences from previously completed projects. Data collected should be recorded together with project characteristics in an accessible storage medium (e.g. company-wide experience database). This step involves a time-consuming data collection process to establish a valid evolution baseline. It is also important to collect data from completed projects whose profile characteristics are representative for future ones in the organization.

5. In reality, no two evolution are identical, just as no two projects are identical. Two evolution occurrences with the same attribute values in the categorization framework are considered "identical". Still, the number of different value tuples of the attributes is so large that we have to define a set of criteria to group evolution occurrences, e.g. in what way two evolution with "minor" difference can be regarded as identical. The grouping criteria can e.g. determine to ignore one or several dimensions in the categorization framework.

6. Retrieve and analyze the evolution data stored in the experience database, grouping them according to the criteria above. Each group of similar evolution with significant number of occurrences (say more than 5% of the total number of evolution being analyzed) can be seen as an evolution pattern, because it represents a type of changes which frequently occur during a particular project.

7. The identified evolution patterns above are then recorded together with the project characteristics in a project history in the experience database for the organization. The project history includes five main parts:

- The characteristics (including e.g. project profile measures) of project in which the evolution patterns above are captured (cf. project characterization framework in section 6.4).
- The actual effort and distribution over phases and activities in development life cycle.
7.7. Empirical Evolution Planning Approach

- The estimation capability index (ECI) which represents the ability to estimate effort of activities in the project (see below).
- Evolution patterns which are recognized. Each of them is associated with an occurrence frequency, impact on schedule, cost and quality based on run-time measurements.
- The total evolution occurrences together with their total cost, evolution density per month and average cost of each evolution occurrence.

8. A more thorough analysis is initiated to identify **empirical relations** between project profile measures and the identified evolution patterns for all completed projects. Such inter-project empirical relations and their associated cost are useful to make an initial plan for a new similar project.

**Estimation Capability Index**

As mentioned above, ECI represents company’s current ability to estimate effort in completed projects. It can be derived by applying following formula:

\[
    \text{EstimationCapabilityIndex} = \frac{\text{ActualEffort} - \text{EstimatedEffort}}{\text{EstimatedEffort}}
\]  \hspace{1cm} (7.1)

A positive ECI means under-estimation, while a negative one is interpreted as over-estimation of actual effort. That is, ECI prefers to be kept as low as possible. The ECI is converged to zero as the estimates become accurate. The ECI can be calculated for either the entire project or individual activities in the development life cycle.

**7.7.2 Initial Planning Support**

A new project is initiated with a specific project/product profile. The project profile measures comprises e.g. project manager, participants, contractual constraints (price, delivery date, requirement stability), customer and eventual external supplier. The product profile contains characteristics such as: operating platform, architecture and language. An initial task network (i.e. project plan) is configured by following steps below (see also figure 7.7):

1. A new project is initiated and characterized according to the identified project profile measures based on existing knowledge at hand. A project **search criterion** which specifies values for those project profile measures, is then created.

2. Assume that one or several similar completed projects are selected from the experience database based on the search criterion. In case there are many similar projects, the search criterion can be further restricted by assigning more values for project profile measures in order to select the best-matching project candidate. If the number of project candidates is still large, the best-matching candidate (also referred to as baseline project) is selected by summarizing the weights associated with each profile measure. It means that the project candidate which has a highest sum, is chosen to be the baseline project.

3. An initial project plan can be automatically generated (by e.g. the Planner). The project plan includes initial cost estimates for the phases or even activities. Such cost estimates are then revised based on the **cost estimation technique** (see section 7.7.4 below).
4. A set of evolution patterns recorded together with the baseline project are then retrieved from the experience database. The project manager utilizes those evolution patterns to configure a more realistic, flexible plan for the current project. The question is basically: how to make the project plan better prepared for the potential evolution which can be predicted from the baseline project? (see evolution pattern analysis in section 7.7.5 below). It is not necessary to carry out both steps 3 and 4. Which step is performed depends on the availability of project information in the experience database.

5. The project plan is now ready to be executed.

![Diagram of Evolution based initial planning support]

*Figure 7.7: Evolution based initial planning support*

### 7.7.3 Iterative Planning Support

Project planning is an iterative process. First we need an initial plan before the project starts. As more information becomes available along with the progress of the project, plan and schedules must be revised. The planning process is only complete when the project itself is completed. It clearly indicates that the project plan may be the most "versioned" document of the project.

We distinguish three distinct situations the project has to deal with during execution:

1. A known and anticipated evolution patterns occurs. In this case, the initial planning has already reserved necessary resources or time to cope with the evolution. In case, such extra effort may not be sufficient, the project schedule can be revised as described in 7.5.2. It is then important to record such a deviation for future calibration.

2. A known but unanticipated evolution pattern occurs. Due to its minor impact or low frequency the evolution pattern is intentionally ignored (see 7.7.5 below). In such a case, the experience database is consulted, and the evolution pattern analysis is performed to estimate the potential impact. The actual effort to deal with the particular evolution is also recorded.
3. An unknown and consequently unanticipated evolution pattern occurs. In this situation, the evolution occurrence is classified by the framework, recorded its actual impacts to the experience database for planning future projects.

To react to an evolution during project execution, the layout (task insertion and removal) as well as the schedule (time, human resource) of the task network may need to be modified. In most cases, those modifications involve human interaction to make effective decisions. The bottom-up process model evolution approaches (in section 7.5) describe how to deal with such modifications. In the following, we present two techniques to support initial and iterative planning steps above.

### 7.7.4 Cost Estimation Technique

Project cost represents a wide spectrum of aspects such as effort spent by its activities, training cost, equipment cost, and so on. Below, we only consider the effort cost to activities during the development life cycle. Such project costs, often expressed in terms of hour or money, are important to negotiate project budget, product price, and to schedule time/resources during the project. During initial project planning, the project cost is usually estimated based on the available knowledge about the project and product at hand. Such estimates about project behavior are elaborated at the time we have least knowledge about project. In our case, the project history of the baseline project helps to determine such cost estimates. Instead of having heuristic rules to determine cost estimates, we distinguish three projection techniques to improve their accuracy.

**Analogy-based projection** The estimate of a new activity is predicted based on the actual effort of the similar activity from completed projects. This technique is simple to realize but hardly provides the most accurate estimates due to inherent differences between projects. An example of such cost projection is presented in the case study in section 9.6.3.2 where we establish a baseline for actual effort distribution over development life cycle.

**ECI-based projection** The project cost is estimated according to the estimation capability index of the baseline project or of all completed projects. That is, the effort estimate of a new activity is mainly elaborated by an initial estimate and estimation capability index of the baseline project. The initial estimates are derived by applying existing estimation method. E.g. the coding activity initially estimated to 120 hours, and the ECI of the baseline project is determined to be +0.3 in the baseline project. A revised estimate ("actual effort") can be found by deriving the ECI-equation in 7.1.

\[
\text{Revised Estimated Effort} = \text{Initial Estimated Effort} \times (ECI + 1) \quad (7.2)
\]

The revised effort is calculated to be 156 hours. The equation 7.2 can be used as a simple algebraic effort prediction model. The quality of the revised estimate for an activity is considerably improved if the ECI in equation 7.2 is derived from the same activity rather than from the entire baseline project. It should also be noted that this technique strongly depends on the accuracy of the initial estimate which is either intuitively elaborated by the project manager, or derived by the existing estimation method. This cost projection technique is discussed further in the case study in section 9.6.3.1.
Contingency-based projection Instead of using the ECI as above, this projection technique is based on the frequency of evolution occurrences and their impacts recorded in the baseline project. An example of such general evolution data is presented in the case study in section 9.6.2 for all studied projects. The initial effort estimates are elaborated by existing estimation method. Such estimates are then revised by adding more time/resources which correspond to the evolution cost recorded in the baseline project. The amount of additional effort (e.g. contingency factor) can be distributed according to the density of evolution (i.e. evolution occurrences per month or in particular development phase). This technique to estimate effort uses the total evolution data from the baseline project, and is thus expected to provide less accurate estimates than a more profound analysis of recorded evolution patterns in section 7.7.5 below.

There are currently two established techniques of software cost estimation in the literature: i) estimation by analogy and ii) algorithmic cost modeling (e.g. COCOMO [Boe81, Boe]). The proposed projection techniques above are inspired by both. The analog- and contingency-based techniques support estimation by analogy because they is based on the historical evolution information from completed and similar projects of the organization. On the other hand, the ECI-based technique derives a cost model whose coefficients are tuned to suit a specific situation of an organization. All three projection techniques utilize the information recorded in the project history of the experience database: actual effort; ECI and evolution data. Recorded evolution patterns are used in the analysis described below.

The cost estimation technique is used to improve the initial plan produced by the Planner. The effort revision of tasks in the project plan (i.e. task network) can be carried out by the approach presented in section 7.5.2.

7.7.5 Evolution Pattern Analysis

A set of evolution patterns are identified and recorded in the project history. Such information is valuable to configure a more realistic and flexible project plan which better anticipates evolution. We present below how recorded evolution patterns can be analyzed to support project plan configuration. We distinguish two kinds of evolution patterns. They are i) evolution patterns which imply changes or adjustment to the generic process models; and ii) evolution patterns which must be anticipated by extra reserved resources or effort (i.e. contingency planning).

When the baseline project indicates a significant representation of evolution patterns of the former type, the top-down evolution approach in section 7.4 is adopted to change generic process models. For example:

- If an evolution pattern indicates intrinsically unstable requirements, we may choose the incremental system development and delivery model to replace e.g. the waterfall model employed in the baseline project. The period of time between deliveries is thus reduced, and several customer feedback iterations are facilitated.

- If an evolution pattern indicates high error rate in source code, we may introduce more rigorous quality assurance activities (e.g. inspection, review, ...) in the development process model to reduce the risk in testing phase. Introduction of e.g. Cleanroom software engineering [MDL87] can be another option.
Sometimes, the baseline project shows a high representation of evolution patterns which do not necessarily imply changes to the generic process model. Rather, they need to be anticipated into the project plan by reserving additional time or resource to affected tasks (i.e. contingency planning). For example:

- If an evolution pattern indicates large turbulence during coding activity, a contingency factor (the percentage of extra time/resources based on the associated impact) should be reserved for possible re-work in implementation phase in the project plan (by schedule revision technique in section 7.5.2).

- Some possible extra activities should be planned and scheduled in advance to deal with an expected evolution pattern during a particular phase. (by task insertion operation in section 7.5.1.1).

In the following, we present two approaches to analyze evolution patterns. The first one takes into account the project history of the baseline project. The second one uses the inter-project empirical relations between project profile and evolution patterns to support effective planning.

### 7.7.5.1 Baseline Project Analysis

We propose a systematic approach to conduct evolution pattern analysis and to make corresponding decisions. As mentioned above, evolution patterns are identified by grouping collected evolution occurrences based on a set of criteria. Such grouping criteria specify rules to determine whether two evolution occurrences are considered as "identical". Usually, this grouping process is simply done by ignoring one or several dimensions in the categorizing framework. E.g. we identify an evolution pattern, called *customer requirement revision* which indicates evolution origin (where), cause (why) and scope (what) without explicitly specifying when, how (reaction) and by-whom it is detected and handled. The recognition process of evolution patterns is described in the case study in section 9.7.3. An evolution pattern can be further elaborated by identifying different change *events*, and each event has alternative *reactions* and *impact* on the schedule/cost/quality. To provide a control mechanism for analysis and decision making, we create and use *evolution scenarios* approach which is inspired by the Riskit approach in [Kon96]. Riskit suggests to classify identified risks into elements (factors and events), and describes plausible reactions for each event, resulting in a risk scenario. But this approach does not exploit experience from completed projects, and only provides decision support during the project execution.

In our case, an evolution scenario is a path from an evolution pattern to one of its events, then to one possible reaction. The evolution scenario is established based on the information recorded in the baseline project, and is used in both at initial planning and during project execution. Using the evolution scenarios, the main steps in evolution pattern analysis are:

1. For each evolution pattern, identify the possible change events; It can be done by investigating in the neglected dimensions in the categorization framework. E.g. the dimensions which are not considered when recognizing the evolution pattern, are now assigned specific category values.

2. For each change event, determine the alternative reactions (e.g. how-dimension);
3. Each path in the above decomposition graph is an evolution scenario. For each evolution scenario, retrieve the actual frequency and impact recorded in the project history of the baseline project.

4. If there are several evolution scenarios, select the ones with high frequency and/or large impact, and make decision to change the generic or enactable (i.e. project plan before start) process model to set appropriate contingencies to anticipate with most likely evolution.

Figure 7.8 shows an example of such evolution scenarios to support planning process. The incremental requirement change evolution scenario represents a significant frequency percentage among requirement change evolution pattern. The new project is therefore encouraged to adopt incremental system development method. On the other hand, the new project should reserve some extra cost and effort to deal with the sub-contractor delay which implies that a new product must be purchased or even a new sub-contractor must be selected.

![Evolution Patterns Diagram]

Figure 7.8: Evolution scenarios for analysis

This approach uses the evolution patterns recorded in the project history of the baseline project. The approach presented in the next section adopts the company-wide, empirical relations between project profile and evolution patterns to support planning.
7.8. Tool Support

7.7.5.2 Inter-project Analysis

In the following, we present an approach to anticipate evolution based on the empirical relations between project profile and evolution patterns. The empirical relations are identified across all completed projects in the organization. An example of such identification is presented in the case study in section 9.7.4. To retrieve such empirical relations requires a great amount of time, experience, and thus needs tool support. Each identified empirical relation is associated with an impact and frequency which are average value of corresponding ones in the involved projects. The approach can be outlined as following.

1. Characterize the new project according to the identified project profile measures/indicators.

2. Examine the recognized empirical relations to determine the most likely evolution patterns based on the identified project profiles. The associated weights of project profile measures are taken into consideration to identify the evolution patterns which likely occur.

3. The relevant evolution patterns are then investigated, analyzed and their actual impacts are anticipated in the project plan.

7.8 Tool Support

In this section, we briefly outline tool support required to realize the approaches above. Such an outline serves as a start point and motivation for the implementation work which are discussed in depth in the next chapter 8. Based on the proposed approaches above, following tool support are found to be necessary.

- **Experience database**: is a storage offering sufficient facilities for storing, retrieving and updating project history. It includes project characteristic, actual effort, ECI, evolution pattern, and evolution cost/frequency.

- **Project characterizer**: is used to characterize project according to its profile and estimates. The tool is configured by the project measures which are determined to be process evolution driver.

- **Evolution Registrar**: is exploited to collect and categorize process evolution according to the categorization framework.

- **Project Selector**: is used to select matching completed projects based on a set of specified project profile measures.

- **Evolution Analyzer**: is used to extract or retrieve experiences from previously completed projects for. This tool is useful in the empirical evolution planning approach.

7.9 Approach Evaluation

We present below an evaluation of the approaches described in this chapter. Such an evaluation will clarify their rationale and justify our research effort. We evaluate the proposed approaches for EPOS according to following features:
**Totality** Process evolution addresses the problem of adapting never-ending changes to software processes in order to improve productivity and product quality. It can be considered to cover two parallel aspects: changes to the real-world processes and to the models (i.e. process support) which represent the former. We propose approaches to manage evolution to three different levels of process models, i.e. generic, enactable and enacting process models. In contrast to the models, the real-world processes are hard to have control over and thus to manage. Therefore, the approach for real-world process evolution is only vaguely specified.

**Flexibility** We distinguish two strategies for evolving process models. The *top-down* evolution approach implies changes to the generic process models, and then immediately or lately propagates to corresponding enactable or enacting ones. The *bottom-up* evolution approach facilitates for local customization of the task network without affecting the generic process models. The local changes are effective during the task network life time, and are possibly propagated to the corresponding generic process models if necessary. While most PSEEs offer the top-down approach, we in addition provide a more flexible bottom-up approaches to evolve task networks in EPOS.

**Integration** The bottom-up approaches includes operations to manipulate the task network layout and revise the task network scheduling. Such operations are inspired by the supports provided by the project management tools. On the other hand, empirical evolution planning approach is motivated by the recognized benefit from experience reuse. All together, those approaches offer an integrated environment in which supports from process and project management are harmonized within a PSEE.

**Validity** The empirical evolution planning approach is strongly based on the previous experience collected from the organization at hand. It thus strengthens the validity and reliability of the approach as well as its expected results.

**Gradual improvement** The empirical evolution planning approach is defined in such a way that the estimates defined in project plan keep improving as long as the project is executed.

**Self-learning** The project history in the experience database is growing over time, and provides a more and more representative and valid foundation for making better estimates.

As a final remark to the application of the approach, one must seek and get full commitment from all management levels. This is perceived as a major prerequisite, since we must invest up-front in the learning and reuse process evolution before tangible results (e.g. more projects deliver on time, with expected product quality and customer satisfaction) are clearly demonstrated. This delay in beneficial effects is typical for reuse-oriented approaches [Kar95].

### 7.10 Chapter Summary

In this chapter, we have presented different approaches to address the issue of software process evolution. Process evolution is characterized as hard to manage, and cover two different aspects: changes to the real-world process and changes to the process support (i.e. models, types). Approach to support the latter has been distinguished into two manners: 1) the *top-down* process evolution approach implies changes to the generic process models (i.e. process schema, types, classes), and either immediately or lately propagates to corresponding enactable or enacting process models (i.e. instances, project plan
or task network in EPOS); ii) the bottom-up process evolution approach which is constituted by basic operations to manipulate task network layout and scheduling. They facilitate changes to the project plan without affecting the generic models. Such changes are effective during the project life time, but can however be packaged into the corresponding generic process models if they appear to be more efficient and adequate than their corresponding generic models. We also claim that the bottom-up approach is more needed than the top-down one because changes to project plan occurs with much higher frequency than changes to the generic process models. Our claim will be formulated as a research question, and validated with a case study in a later chapter. On the other hand, evolution of the real-world processes are difficult for a PSEE to have control over. The associated approach is therefore specified in a vague manner.

We have also in this chapter presented an approach to support planning, called empirical evolution planning. This approach includes three major steps: i) establishing evolution baselines; ii) supporting initial planning; and iii) supporting iterative planning.

1. The first step is crucial to construct evolution baselines on which the next two steps mainly rely. The evolution baselines are established based on our evolution categorization framework and project characterization framework (in chapter 6). The evolution patterns are identified and recorded together with the project profile to constitute a project history in the experience database. In addition, relevant empirical relations between project profile and evolution patterns across completed projects are also determined.

2. The second step supports to make an initial project plan. The project plan which currently generated by EPOS’s Planner, is modified to anticipate potential changes before project execution.

3. The third step assists to revise project plan during project execution taking into account the unanticipated evolution occurrence.

To support the empirical evolution planning, we have define a technique for cost estimation evolution pattern analysis. The cost estimation is consists of three different techniques: i) analogy-based; ii) ECI-based and iii) contingency-based projections. The evolution pattern analysis includes a baseline project and inter-project analysis. The former relies on evolution patterns recorded in the project history of the baseline project. The latter uses the empirical relations between project profile and evolution patterns across completed projects to support planning.
Chapter 8

Implementation Issues

8.1 Introduction

In this chapter, we describe how the proposed evolution categorization framework (in chapter 6) and approaches to manage process evolution (in chapter 7) are implemented in EPOS. The experience database in EPOS is first described in section 8.3 followed by the description of how different approaches to manage enactable and enacting process models are implemented in section 8.4. Section 8.5 presents tools providing planning support based on the empirical evolution planning approach (in section 7.7). Brief descriptions of an experience database prototype for the studied software company and a World Wide Web based experience database are presented in sections 8.7 and 8.8 respectively.

8.2 Requirements for Implementation

Below, we outline some major requirements for our implementation work in EPOS. They will justify the solution decisions we arrive at during the presentation in subsequent sections. The requirements are among others:

Conformance The extended features in EPOS should correctly reflect the functionalities described in the framework (chapter 6) and approaches (chapter 7). That is, the characteristic features of the categorization framework, e.g. configurable, flexible, etc; and of the approaches, e.g. operational, integrated, etc, should be completely present.

Extendibility The selected solution choices should be easily extended in the future to adapt to new changes and needs. That is, both the extension in data models and new tools should be open-ended and upgraded without great effort.

Integration The new extensions with their associated tool support should be integrated with the existing EPOS environment. That is, existing and new tools must effectively work together within an integrated environment.
8.3 Experience Database in EPOS

In this section, we present in details the implementation issues for our experience database in EPOS. As mentioned in chapter 7, the experience database comprises a collection of project histories from completed projects and associated tools to manipulate them. The project history includes five elements: i) project characteristics; ii) estimated and actual effort for activities; iii) Estimation Capability Index (ECI) iv) evolution occurrences; and v) evolution patterns. The associated tools needed to record, access and analyze such information are also briefly outlined in section 7.8. In the following, we describe how the project history is stored by extending the existing EPOS’s data models, and how the associated tools are used to manipulate them in EPOS.

8.3.1 Project Characterization

In section 6.4, we presented a framework to characterize project according to its profile, estimates and outcomes. The aspect of project profile was then elaborated further to identify a set of measures which most likely influence process evolution. Those measures which was depicted in figure 6.3, represent three areas: organization, process and product. In this section, we will show how those project profile measures are modeled in EPOS. Figure 8.1 depicts new types which are defined to model project profile in EPOS.

![Diagram of EPOS-PM types for project profile]

*Figure 8.1: New EPOS-PM types for project profile*

A project connects to its profiles by relation has_profile. An abstract profile type is subtyped to three other abstract subtypes representing three identified areas: organization, process and product. The profile measure types (i.e. customer, constraint, platform …) are all instances of EPOS’s entity...
type, and are allowed to have values which are defined in figure 8.2 below. We define those new types and structure them in such a manner that the data model can be extended in the future due to new gained knowledge without affecting the associated tools. That is, the number of subtypes of org-profile, proc-profile and prod-profile can be enlarged or confined without changing the Project Characterizer (described below).

Project Characterizer Tool

Figure 8.2 shows a screen picture of the Project Characterizer tool in EPOS. This tool is used to characterize project and store profile measures in the EPOS experience database according to the data model presented above. Such characterization process is usually carried out at the project start. Note that the project measures and their values which are included in this tool, are defined with a clear intention for the case study against a software company (see more in chapter 9).

The Project Characterizer provides support to characterize a new project as well as to display the recorded characteristics of a particular one. The profile measures and their values can be defined or configured by editing a text file. The Project Characterizer reads such a configuration file before it is opened. For each profile measure which is defined in the configuration file, a corresponding type is created in the data model (as shown in figure 8.1). The profile measure names and possible values are then incorporated to build the user interface of the Project Characterizer. Therefore, when a new profile measure is added to the configuration file, a new type will automatically be created, and the Project Characterizer will then display the new profile measure when it is invoked. The configuration file for profile measures associated to the Project Characterizer has e.g. following format in Prolog (represented as Prolog facts).

Org-profile
   Customer(string),
   Customer type(Bank, Internal, External, Joint development),
   Supplier(string)).

Proc-profile
   Constraint(
       Price(Fixed, Negotiate-able, Fully funded, Own risk),
       Delivery(Fixed, Negotiate-able),
       Requirement stability(Frozen, Revisable))
   Project type(New development, Upgrading, Maintenance, Other),
   Competence(Non-existent, External existent, Training needed, Internal available),
   Technology(New, Internal unknown, Partly known, Well-known),
   Difficulty(Minor, Middle, Manageable, High)).

Prod-profile
   Server platform(....),
   Server language(....),
   Client platform(....),
   Client language(....)).

There are three levels which are represented by the different indentations in the configuration file. The first level is interpreted as subtypes of the profile type. Profiles in the second level are defined as
Figure 8.2: Project Characterizer

**Project Information**
- **Project number:** 5079
- **Project name:** MICOSAT
- **Product number:** 24987
- **Product name:** MICOS

**Customer:** Fokus Bank
- **Customer type:**
  - Bank
  - Internal
  - Joint development
  - External
  - Joint development

**Supplier:** Microsoft
- **Estimated cost:** 1.600.000
- **Estimated effort:** 6500 hours
- **Estimated duration:** 15 months
- **Effort distribution:**
  - Planning: 1%
  - Development: 99%
- **Milestone/Deliverable:** Requirement specification
- **Date:** 01 April 199X

**Contractual constraints**

**Price:**
- Fixed
- Negotiable
- Fixed
- Negotiable

**Delivery:**
- Fixed
- Negotiable
- Reusability

**Function point:**
- Number of DB tables: 25
- Number of UI screens: 10
- Number of produced reports: 5

**Product Information**

**Developed KLOC:**
- 1
- 5
- 10
- 50
- 100
- More

**Generated KLOC:**
- 1
- 5
- 10
- 50
- 100
- More

**Server platform:**
- MS-Windows
- Windows-NT
- DB2
- Oracle
- Sybase
- Progress
- Unix

**Client platform:**
- MS-Windows
- Windows-NT
- DB2
- Oracle
- Sybase
- Progress
- Access

**Server language:**
- C
- C++
- Smalltalk
- Cobol
- Progress
- Visual Basic
- Oracle 2000

**Client language:**
- C
- C++
- Smalltalk
- Cobol
- Progress
- Visual Basic
- Oracle 2000
subtypes of those in the first level. Information in the third level is defined to be attributes of the type in the second level. The possible profile values are listed in parenthesis. The project estimates on cost and duration and product estimates on e.g. sizes are fixed fields in the current implementation of the Project Characterizer. Their values are stored as attributes of the particular project and product. When the configuration file for project profile measures is modified, the layout of different fields in the Project Characterizer is not as tidy as depicted in figure 8.2. That is the price we have to pay for having both flexibility and user-friendliness.

The main features of the Project Characterizer Tool can be summarized as following:

**Characterization** The tool is used to characterize the project according to the project profiles and original estimates. The project profile measures and their values are flexibly configurable, while the project and product estimated measures are fixed. Such configuration of project profile measures is done by manipulating the textual configuration file.

**Presentation** The tool also facilitates for displaying the recorded characteristics of a particular project. Such a feature is useful when examining completed projects for similarities. However, we also provide automatic support for selecting similar completed projects on the basis of specified values of project profile measures (see more in section 8.5.1).

### 8.3.2 Process Evolution Categorization

In the following, we present how the evolution categorization framework is applied to classify and record evolution occurrences in EPOS experience database. Figure 8.3 depicts the necessary extension of the existing data model to store evolution and evolution pattern of a particular project.

![Figure 8.3: New EPOS-PM types for evolution categorization](image-url)
A project is associated with a set of evolution occurrences which is categorized by having a relation with a category. Identified evolution patterns are also recorded in the same manner but with another relation from a project. An evolution type has attributes to store estimated and actual cost together with a textual description of the particular evolution occurrence. The dimension, aspect and category types which have an attribute value, are used to build a categorization tree (see below). We take into account the categorization framework which is constructed for the studied company (see in figure 6.2) to implement in EPOS. The dimension, aspect and category values in the framework are then described in a textual configuration file which has a following Prolog representation (only partly shown).

where('Project internal', 1).
where('Executive', 1).
where('Quality manager', 2).
where('Steering committee', 3).
where('Project manager', 4).
where('Project member', 5).

where('Project external', 2).
where('Customer', 1).
where('Sub-contractor/Supplier', 2).
where('Other', 3).
where('Competitor', 1).
where('Market', 2).
where('Technology', 3).

.... etc...

A statement in the configuration file is always started by a dimension value (i.e. where, why, what, when, how, and by-whom). There are two different types of statement which are distinguished by its number of parameter. The statement having two parameters represents aspect, while category value is specified in the statement with three parameters. The numeric parameters are used to determine the membership and ordering of category in different aspects.

Based on the information in the configuration file, six categorization trees are established representing six dimensions in the categorization framework (i.e. where-, why-, what-, when-, how-, and by-whom-tree). Such a categorization tree has a dimension instance as a root (e.g. where-dimension), and is then decomposed into a set of aspect instances (e.g. project internal, project external and other). Each node of aspect instance is then linked to different category instances with corresponding values. Figure 8.4 depicts an example of a categorization tree for where-dimension. The categorization trees are fixed as long as the configuration file remains unchanged. The eventually new changes in the configuration file are incorporated in the categorization trees at the project start. That is, each project has opportunity to configure the categorization framework by itself. The existing categorization trees are then modified (i.e. creating new instances or removing existing ones) according to changes to the configuration file. In that way, the categorization framework is flexibly configured to reflect new needs or insights during its usage.

In the following, we will illustrate how such categorization trees are used to perform operations such as: recording a new evolution occurrence; searching a particular evolution; and navigating among a set of evolution occurrences associated to a project.
8.3. Experience Database in EPOS

Figure 8.4: An example of a categorization tree

Recording a new evolution occurrence

A new evolution occurrence of a project is categorized by first assigning values for category in the categorization framework. A new instance of evolution type is then created and linked to those selected categories by categorized_by relationship. The given estimated cost and a description of evolution are stored in the new evolution instance which is finally linked to a particular project instance by occurs relationship.

Searching a particular evolution

This operation is useful for the Analyzer Tool (see figure 8.12 below) to retrieve e.g. the frequency as well as the impact of a particular evolution occurrence. A search mask is determined by assigning values for the categories in the framework. It is allowed to specify don’t care values for one or several categories in the search mask. The search operation is then described by the following pseudo-code.

For each selected category do
begin
  Make a triple value DAC = [dimension, aspect, category] where
dimension and aspect are parent nodes of category in the category tree. (1)
  Retrieve a set EvolSet consisting of evolution instances which are:
  linked to the category by categorized_by relationships (2)
  and linked to the particular project by occurs relationships.(3)
The result is a collection of the evolution instances which appear in all collected sets EvolSets. (4)

The above pseudo-code is still valid if we want to investigate all recorded evolution occurrences regardless which project they belong to. It is achieved by ignoring the test in (3). The search mask can also comprise aspect values (e.g. Project internal in where categorization tree). It means that we want to retrieve all evolution occurrences which are caused by all categories belonging to the given aspect. In that case, the statement (1) will make as many DACs as there are categories under the selected aspect value.

Navigating among evolution occurrences

This operation is relevant when we want to browse among previous evolution occurrences. Usually, we only estimate the impact of the evolution occurrence at the first time. Therefore, this navigation facility is necessary to assign the actual impact on cost for recorded evolution data. Evolution occurrences to a project are organized and enumerated in the same order as they are recorded. That is, they can be browsed back and forth, and their values can be modified simultaneously. The navigation algorithm has an opposite execution sequence comparing to the search one described above. We start with an evolution instance to a particular project. For each categorized by relationship from the evolution instance, we first retrieve the category and then create the triple value DAC by searching up the corresponding categorization tree. A set of collected DACs are then sent to the Evolution Registrar Tool (see below) for displaying.

Evolution Registrar Tool

Figure 8.5 depicts the screen picture of the Evolution Registrar Tool in EPOS. This tool is used to record evolution occurrences to a particular project according to the recording algorithm above. The Evolution Registrar Tool is also used to navigate between recorded evolution occurrences to the same project as described above.

Similar to the Project Characterizer Tool, the aspect and category values in the Evolution Registrar Tool are specified in the configuration file. In that way, the tools are more flexible and easily adapted to changes without great modification effort. It should also mention that those tools have been actively used to record project information as well as evolution data which are collected from the case study. The EPOS experience database is currently stored with those data for further analysis.

8.4 Implementation Issues for Task Network Evolution

In this section, we present the implementation issues for the approaches to evolve enactable and enacting process models (i.e. task network) in EPOS. Section 7.5 describes two approaches: 1) manipulation of the task network layout; and ii) revision of the task schedules and resources. Both approaches can be performed to the task network before and during enactment.

In the current implementation of EPOS, the Planner constructs the task network by both creating its task instances in the EPOSDB and displaying them graphically. To be able to manipulate with layout
as well as scheduling information of the task network before it is executed, we store the generated task network in a cache without actually creating its task instances in the EPOSDB. Such temporary task network which is called a proposed task network, is simply represented by nodes (or objects) in our graphical package XPCE\(^1\). The task instances of the proposed task network are actually created in the EPOSDB upon explicit user demand. It means that the proposed task network gives us opportunity to change the task network layout and task scheduling before enactment. Of course, the task network layout manipulation and task schedule/resource revision approaches are also applicable for the real task network, i.e. during enactment. In the following, we therefore discuss how those approaches are implemented in EPOS without distinguishing the real or proposed task network.

8.4.1 Manipulation of Task Network Layout

In section 7.5.1, we have presented two basic operations to manipulate the task network layout. They are task insertion and task removal operations. To do so, the TaskEntity type, the Planner and Process Engine are required to be modified. The type-level attributes of TaskEntity which are affected by task insertion or removal operations, must be re-defined at the instance-level. Such double representation is needed to separate and encapsulate changes to the instances. We therefore add following new instance-level attributes (i.e. PRE_STATIC, POST_STATIC, PRE_DYNAMIC, and CODE) for TaskEntity type as follows (expressed in SPELL). The names of new instance-level attributes are prefixed by inst_.

```plaintext
entity_type(
    [name(taskentity),
     subtypeof(pm_entity),
     inst_attr([name-inst_staticpre, domain-prologterm(200),
                  default-[]),
     inst_attr([name-inst_staticpost, domain-prologterm(200),
                  default-[]),
     inst_attr([name-inst_dynamicpre, domain-prologterm(200),
                  default-[]),
     inst_attr([name-inst_code, domain-prolog(3000),
                  default, []]),
     ......
]
```

The type-level DECOMPOSITION and FORMALS are currently represented by relations decomptype and formalparms respectively. The former connects between two task types, while the latter links a task type to its input or output product types. Those two relations are mainly used by the Planner for task network construction (i.e. vertical break down and horizontal chaining). When the task network is executed, two other equivalent relations, but at instance-level, subtasks and actualparms are used to connect between corresponding instances. Those instance-level relationships which are used by the Process Engine for execution, are independent of the DECOMPOSITION and FORMALS type information. The problem is how to make the Planner to separate new changes done to the instance-level. To remedy the problem, we add an extra attribute effect to the relation types decomptype and formalparms. The attribute has three possible values: none (default), insertion and removal.

\(^{1}\)Jan Wielemaker, Social Science Informatics, University of Amsterdam, The Netherlands.
When a task insertion operation is performed, new decomptype and formalparams relationships are created with insertion effect. When a task is removed, the affected decomptype and formalparams relationships are not deleted but their effect attributes are assigned value removal. The Planner will then reason about the relationship whose effect value is either none or insertion, and ignore those with removal values. The SPELL-definitions of those two relations with new effect attribute are now:

```plaintext
relation_type(
    [name(decomptype),
     subtype(object),
     rolecard(source-task_td,'0'-n, dest-task_td,'0'-n),
     inst_attr([name-effect, domain-string(16), default-none])
     ....
)

relation_type(
    [name(formalparams),
     subtypeof(object),
     rolecard(source-task_td,'0'-n, dest-data_td,'0'-n),
     inst_attr([name-effect, domain-string(16), default-none])
     ....
)
```

The Planner and the Process Engine are also modified to react correctly when the task network layout is changed. The Process Engine must use the new instance-level attributes (i.e. PRE_DYNAMIC, CODE) to interpret and execute the task network when they are non-empty. Similarly, the Planner is enforced to use the new non-empty instance-level attributes (i.e. PRE_STATIC, POST_STATIC) to horizontally expand or vertically decompose high-level tasks. Only the DECOMPOSITION and FORMALS relationships with either insertion or none effect attributes, are used by the Planner.

By performing those changes above, the task network layout can be manipulated (i.e. inserting and removing tasks) without affecting the corresponding types. The Planner and Process Engine are still working with a minor modification. However, when the project (i.e. EPOSDB transaction) is completed, the user is asked to decide what to do with the new changes to the instances. We have defined two Prolog predicates to take care of this.

- **package_change**: The predicate will incorporate all new instances changes to the corresponding types. Specifically, those non-empty instance-level attributes to TaskEntity (i.e. staticpre, staticpost, dynamicpre, code described above) overwrite the existing corresponding ones in the type definition. The decomptype and formalparams relations whose effect attributes are insertion, are changed to none. The decomptype and formalparams relationships whose effect attributes are removal, are deleted.

- **discard_change**: This predicate will remove all decomptype and formalparams relationships whose effect attributes are insertion. The effect attributes whose values are removed, are changed to none. We assume that the same task is not first inserted and then removed afterward.

In the following we show with screen pictures how task insertion and removal operations are implemented in EPOS.
8.4.1.1 Task Insertion Operation

Section 7.5.1.1 describes in details constraints as well as the algorithm how to add a new task to the task network. Figure 8.6 depicts a screen picture of a task network in EPOS before the task insertion operation is taking place. From the pop-up menu of a task node (e.g. cedit in the example) we can decide to add a new task. A dialog box is prompted asking for the type of the new inserted task. In this way, the user can only select one of the existing types in the current project (e.g. review is selected). The location of the new task is determined by pushing the Before or After buttons in the dialog box. Selecting Before or After means the new task is added before or after the task the insertion operation is invoked. The request is then evaluated according to the location constraint of the task insertion operation, and the FORMALS of review task type is examined to determine whether its inputs and outputs types match to the corresponding ones of cedit and ccompile. If the request is allowed the new review task is inserted between cedit and ccompile. The matching product (i.e. test.c) is automatically linked. The new task network is then similar to the one in figure 8.8. The non-matching output (i.e. humanfb) of review is explicitly designated to cedit by the user.

![Task insertion operation in EPOS](image)

The task network is connected but it is still semantically incorrect. A notification box (in figure 8.7) is automatically prompted to notify the user about the inconsistencies in cedit. In this way, the user is encouraged to correct those affected attributes. The FORMALS and DECOMPOSITION information are automatically updated, while other attributes (i.e. PRE_STATIC, POST_STATIC, PRE_DYNAMIC and CODE) must be manually correct to reflect the new semantics. Especially, the CODE part must be extended to utilize the new humanfb input from review task. When Update button is clicked the new information is stored in the new instance-level attributes described above. An equivalent notification box for ccompile is also automatically prompted.
8.4. Implementation Issues for Task Network Evolution

![Diagram](image)

Figure 8.7: An example of prompted notification box for cedit

### 8.4.1.2 Task Removal Operation

Section 7.5.1.2 describes the constraints and algorithm of the task removal operation. Figure 8.8 illustrates a screen picture in which a review task is removed from the task network in EPOS. The user gets a warning about the consequences of such operation, and is asked to confirm the removal operation. The location constraints as well as the FORMALS of the deleted task are evaluated to determine whether the removal operation is allowed. The matching product (i.e. test.c) is then joined between the preceding and subsequent tasks. The superfluous product (i.e. humanfb) is automatically removed. Similar to the task insertion operation, two notification boxes (in figure 8.7) for cedit and ccompile are automatically prompted to enforce the user to correct possible inconsistencies, and new information are stored in the corresponding instance-level attributes. The DECOMPOSITION and FORMALS information are on the contrary updated automatically after the removal operation.

### 8.4.2 Manipulation of Task Scheduling

In section 7.5.2, we have presented approach how to revise the schedule as well as resource associated to the task network before and during enactment. The two screen pictures (figures 8.6 and 8.8) in the previous section only depict the proposed task network (i.e. graphical task nodes are shown with a white background). In the following, we will demonstrate how task schedule and resource are revised during enactment (i.e. the graphical task nodes are filled with gray color).

### 8.4.2.1 Definition of Time Attribute to TaskEntity

In section 7.5.2, we also identified some extensions needed to be done to the task model, specifically to the TaskEntity type. A new instance-level attribute effort for TaskEntity is therefore defined as a Prolog list of triple [Starttime, Elapsetime, Duration]. A new triple (S,E,D)
Figure 8.8: Task removal operation in EPOS

is created whenever the task schedule is revised. Thus, the first entry of the effort attribute list represents the estimated effort value for the particular task. In addition, when the task is completed, the last entry in the effort attribute list represents its actual effort. The estimation capability index (ECI) which is calculated by the ratio of estimated and actual effort (c.f. equation 7.1 in section 7.7.1), can be derived by the first and the last entries in the effort attribute list. The additional part of TaskEntity definition in SPELL looks as follows:

entity_type(
    [name(taskentity),
     subtypeof(pm_entity),
     inst_attr([name-effort, domain-prologterm(100), default-[[0,0,0]]),
     .......

In EPOS the task is executed as soon as its preceding tasks are completed and its PRE_DYNAMIC is evaluated to true. Such execution strategy is still valid and prevailing even though the Starttime value is associated to a task instance. That is, the value of Starttime is just a planned figure and does not influence when the task is actually executed. If a task is delayed and not finished in time, the Starttime values associated to subsequent tasks are then adjusted accordingly. Such adjustment is done by storing a new triple (S,E,D) in the effort attribute.
8.4.2.2 Revision of Task Schedule

The schedule of a task can be changed by invoking the Change schedule entry of the pop-up menu attached to the given task (see figure 8.8). The Schedule Revision dialog box is then prompted as depicted in figure 8.9 to display the existing values for start time; elapse time and duration attribute of the selected task. Those values can then be modified by the user. As mentioned in section 7.5.2, certain attributes are allowed to be changed depending on the value of elapse time attribute at the moment the operation is invoked. The rule can be summarized as follow:

**If** elapse time = 0 **then**
- start time and duration can be changed

**Else if** elapse time < duration **then**
- duration can be changed

**Else if** elapse time = duration **then**
- all three can be changed;
- elapse time := 0 / rework */
- duration := duration + extra duration

These rules are taken care of by making the text fields in the dialog box possible or impossible to be edited. The new entry of triple (S,E,D) is created and concatenated to the instance-level cost attribute list of the task (see section 8.4.2.1).

![Schedule and resource revision in EPOS](image)

Figure 8.9: Schedule and resource revision in EPOS

After such task schedule revision, the schedules of other dependent tasks should be adjusted accordingly. Unfortunately, we do not have sufficient time to implement such automatic feature due to inher-
ent complexity in determining the sequencing through tasks in the task network. The automatic initial scheduling which should be performed by the Planner has not been implemented yet. We rather prefer to use the experience data to schedule manually as described in section 8.5 below.

8.4.2.3 Revision of Task Resource

Figure 8.9 also depicts how to replace a current performer of a task by another participants which are listed in a separate dialog box. One or more persons can be assigned to the same task. In addition, the selected task can also be suspended because the intended performer is not available. In that case, the triple (S,E,D) is temporarily frozen, and is restarted when the task is explicitly resumed. The selected task can also be canceled because it is not relevant any more for the project. Two new task states are therefore introduced and the state transition diagram for task state in figure 4.10 is augmented as depicted in figure 8.10.

![State Transition Diagram]

Figure 8.10: Two new task states and extended state transition diagram

Naturally, the task schedule in the task network is affected by suspending or canceling a particular task. The resource is better and effectively utilized by if it can be optimally allocated. As said above, automatic support for adjusting task schedule after such operations is not offered by the current implementation of EPOS.
8.5 Tool to Support Empirical Evolution Planning

In section 7.7, we have presented an approach to support planning based on empirical evolution. It is consisted of three major steps: i) establishing evolution baselines; ii) supporting initial planning; and iii) supporting iterative planning during project execution. In the following, we describe how those steps are supported by tools in EPOS.

The evolution baselines are established by collecting evolution data from completed projects. The Evolution Registrar Tool (in figure 8.5) is used to categorize and then record evolution occurrences during projects. In parallel, the project is characterized by the Project Characterizer Tool (in figure 8.2). Both the evolution data and project characteristics are recorded in the EPOS experience database which is described in section 8.3. The extension in TaskEntity in EPOS to represent time-related information is also useful to record both estimated and actual project cost. On the basis of such information the Estimation Capability Index (ECI) for the entire project as well as for each individual task can be derived. In general, we have managed to provide adequate tool support to establish baselines for evolution, effort estimation and project characteristics.

We have implemented two tools to assist the empirical evolution planning support. Figures 8.11 and 8.12 show the screen pictures of the Project Selector Tool and Evolution Analyzer Tool. Their features are described in the following sections.

8.5.1 Project Selector

The Project Selector is used to select completed projects which are matching with the assigned profile measures. The tool uses the same project profile measures which are defined in the Project Characterizer (figure 8.2). It also means that the project profile measures are defined by the same textual configuration file in section 8.3.1. It is not mandatory to specify all values for the project profile measures in the Project Selector to search for matching completed projects. Such relaxation is useful at the moment when the knowledge about the new project is somehow scarce. The assigned profile measures are used to query against the EPOSDB to retrieve matching projects. Only projects whose characteristics completely correspond to the query, are reported in the result box. The number of matching projects can be reduced by assigning values for more profile measures. The weights associated to a particular project profile measure are not taken into consideration in the current implementation of the Project Selector Tool. The Project Selector Tool is not properly tested with a large amount of project information. This is due limited project information to be collected from the software organization in the case study.

8.5.2 Evolution Analyzer

The Evolution Analyzer is used to retrieve information about frequency as well as cost related to evolution. This tool can analyze amount of evolution data from both a single project and several projects. The tool also applies the same categorization framework constructed for the studied company and depicted in figure 6.2. It has three main functionalities.

Evolution pattern discovery In section 7.7.1, we present approach to establish evolution baseline. It requires among others to identify appropriate criteria to group a large number of evolution
Figure 8.11: Project Selector

Matching Project number: [5073, 5645]
Matching Project name: [MICOSANT, HOKUS]

Project type:
- [ ] New development
- [ ] Upgrading
- [ ] Maintenance
- [ ] Other

Life-cycle model:
- [ ] Waterfall
- [ ] Prototyping
- [ ] Spiral
- [ ] Own model

Contractual constraint:
- [ ] Fixed
- [ ] Negotiable
- [ ] Fully funded
- [ ] Own risk

Delivery:
- [ ] Fixed
- [ ] Negotiable

Requirement stability:
- [ ] Frozen
- [ ] Revisable

Server platform:
- [ ] MS-Windows
- [ ] Windows-NT
- [ ] OS/2
- [ ] Unix
- [ ] Oracle
- [ ] Sybase
- [ ] Progress
- [ ] Access

Server language:
- [ ] C
- [ ] C++
- [ ] Smalltalk
- [ ] Cobol
- [ ] Progress
- [ ] Visual Basic
- [ ] Oracle 2000

Client platform:
- [ ] MS-Windows
- [ ] Windows-NT
- [ ] OS/2
- [ ] Unix
- [ ] Oracle
- [ ] Sybase
- [ ] Progress
- [ ] Access

Client language:
- [ ] C
- [ ] C++
- [ ] Smalltalk
- [ ] Cobol
- [ ] Progress
- [ ] Visual Basic
- [ ] Oracle 2000
occurrences to relevant patterns. This tool doesn’t offer automatic identification of such evolution patterns. Such support requires a great deal of intelligent reasoning about and complex processing on evolution data. By using the Evolution Analyzer, the user can however adopt a trial and error approach to discover the evolution pattern which exceeds the upper edge of frequency. We can start to query by assigning all values for all aspects. If the frequency is still low and far from a critical value, one or several aspects can be ignored in the query by selecting the Do not care value. By trying this way, we can occasionally discover critical evolution patterns which can be recorded together with the project.

Contingency determination The Analyzer Tool can also be used to determine the cost of a particular evolution pattern recorded in the baseline project. This actual evolution impact on cost is useful when the project plan is initially elaborated. That is, an appropriate amount of resource or time is anticipated or reserved in the plan to react to the given evolution pattern (contingency planning). This functionality is also important for the cost estimation technique which is based on the total evolution impact on cost of the baseline project (described in section 7.7.4).

Evolution cost estimation The project plan is continuously subjected to revision during project execution. Sometimes, a particular evolution pattern which is not taken into account in the project plan, but actually occurs. The Analyzer Tool can then be used to query the actual cost of similar evolution occurrence from both the baseline project and all completed projects in the experience database. Such actual evolution cost is helpful to determine the cost estimates of a similar evolution occurrence at hand.

The Evolution Analyzer can be used to retrieve necessary information to establish the evolution scenario (described in section 7.7.5). An evolution pattern for a project can be displayed by using the browsing facility of the Evolution Registrar Tool. The evolution pattern which often ignores one or several aspects in the categorization framework, is then transferred to the Evolution Analyzer Tool for further analysis. The different change events and reactions together with their frequency and impact can be identified by assigning more category values.

8.6 Integration of new tools into EPOS

In summary, the Project Selector, Evolution Analyzer together with the experience database in EPOS have provided adequate support for planning new projects based on empirical evolution data. Still, there are space for improvement in our implementation. We have however managed to demonstrate the usefulness as well as the importance of such empirical evolution data, and make an operational environment to effectively reuse them in planning new similar projects. Figure 8.13 depicts the interaction between the previous process tools in EPOS and the new ones which are introduced in this chapter.

The task network is constructed by task and product instances which are defined in the process schemas (i.e. types, models) in EPOS. Such process schemas are mainly manipulated and managed by the Schema Manager and Task network Editor. Figure 8.13 also depicts the extension of data model in process schema to represent project experience, including e.g. project characteristics and evolution information (see in section 8.3). Project histories are thus constructed by those new types to represent specific experiences of completed projects. The Planner takes into account the existing process knowledge and
Figure 8.12: Evolution Analyzer
specific project goal to construct a task network (i.e. enactable process model). The Process Engine will then interpret and enact the task network to assist and guide the process performance. Both before and during task network enactment, the Project Manager Tool offers supports for manipulating task network layout and for revising task scheduling. The Project Manager Tool also invokes the Project Characterizer Tool at project start to determine project characteristics and store them in corresponding project history. Evolution occurrences are classified and recorded in appropriate project history by the Evolution Registrar which also collaborates with the Evolution Analyzer to recognize evolution patterns. The Evolution Analyzer retrieves recorded experiences from project histories to support Planner to elaborate initial plan. The analytical data from the Evolution Analyzer is also useful for project manager to adjust the project plan (i.e. task network) before and during enactment.

8.7 Example of a Separate Experience Database at XXX

In this section, we briefly present the experience database and its associated tools which is implemented at the case study software company XXX. The details of the case study is described in chapter 9. When we start to conduct the case study, we manage to convince the management at the software company about the necessity of a local experience database in which company’s confidential and sensitive information are stored. A student project has been assigned and supervised by the author to implement such an experience database prototype for the company, called EDBN. EDBN is initially operating on MS-ACCESS database platform and providing three main features:

Measure collection EDBN offers a tool to collect and record a set of existing measures at the com-
pany in the MS-ACCESS database. Those measures are previously collected by filling in various paper-based reports during project life time. The EDBN provides facilities for recording measures such as: spent/remaining work hour and degree of completion for each activity or phase; number of error found in inspection, information on project personnel, .... The information which is stored in the database, replaces the large number of reports being produced monthly by the project managers.

**Information search** EDBN offers a search mechanism towards the experience database. The search is basically initiated by assigning values for project profile measures such as name of project manager, project staff, project type, etc. A set of matching completed projects are then retrieved from the experience database. It is then possible to have direct access the recorded information from those completed projects. Such information can either be printed out or displayed in the screen.

**Report generation** However, the company can not totally converted to a paper-less environment. Monthly status report, progress report and final report for project can easily be generated by the EDBN. Such a feature has reduced amount of required documentation effort and increased the accuracy of information being reported.

The EDBN has been later upgraded to incorporate our categorization framework and record collected project data from the case study. The detailed categorization framework is first elaborated together with the company. The resulting framework is depicted in figure 6.2. The new prototype, called EDBN++, is operating upon ORACLE database platform providing richer supporting features and multi-user environment. Some main extensions can be summarized as follows:

- Incorporating the constructed categorization framework depicted in figure 6.2 to record and classify process evolution during projects.

- Integrating new useful project profile measures (e.g. those depicted in figure 8.1 above) into the existing data schema to better search for similarity between projects.

- Separating the database attributes relating to project characteristics and evolution to make them maintainable.

- EDBN++ supports multi-user access and access control by password.

The screen shots from EDBN and EDBN++ can not be provided here because of two reasons. Firstly, the user interfaces are implemented in Norwegian which is not easily understood for the reader. Secondly, the studied company is not willing to publish their products with fear for being recognized. However, we can safely reveal that the contents of the user interfaces are not far from the ones depicted in figures 8.2 and 8.5 under the experience database in EPOS. Unfortunately, neither the EDBN nor EDBN++ have been actively used to collect project and evolution data, although they are more or less operative. It however demonstrates the resistance and little susceptibility to new changes in the real-world processes. New activities are currently planning at XXX to convert the experience database to WWW-based user interface with local net. The ambition of its usage is also confined in order to get acceptance among personnel.
8.8 A WWW-based Experience Database Prototype

We recognize the fact that EPOS experience database prototype has a limited disseminating effect. We have therefore decided to exploit the new World Wide Web (WWW) technology to spread out our ideas, especially the evolution categorization framework. The author has supervised two diploma students to implement a WWW-based experience database prototype in which the categorization framework is used to classify and record evolution occurrences. The ORACLE database is used as underlying data storage for the WWW-based experience database which is essentially implemented in Java and JFactory development environment from Rouge. The main features of this experience database can be summarized as follows:

**Project characterization:** All the project profiles measures defined in figure 8.1 are incorporated in the tool to characterize projects.

**Process Evolution:** The constructed categorization framework is also defined and used to record and classify process evolution occurrences during projects.

**Analysis and Reporting:** Information of completed projects can be retrieved based on a set of specified project profile measures.

**Dynamic Extension:** The number of measures, including the project profile measure and categorization framework, can be dynamically extended during execution. The corresponding database schema and user interface are then updated accordingly.

Figure 8.15 depicts a screen shot in which an evolution occurrence can be categorized and recorded within a project. The Category and Source in the figure correspond to the Aspect and Category in our framework respectively. This is due to misunderstanding of the students. Figure 8.15 depicts a window in which the cost for an evolution occurrence is estimated and then actually determined. The difference or deviation is then calculated. The evolution cost is expressed in terms of money and working hours. Figure 8.16 shows a window in which the particular evolution occurrence and its corrective actions can be further specified.

The WWW-based experience database is built on top of a trial version of ORACLE database which has currently only limited license at our University. A few improvement works still remain before it can be publicly released.

8.9 Chapter Summary

In this chapter we have presented the implementation issues for the evolution categorization framework and approaches to manage process evolution in EPOS. We have first described the EPOS experience database which includes project history and a set of associated tools to manipulate (i.e. record, access, analyze) experience information. The project history is consisted of: project characteristics; project actual cost/effort; estimation capability index; evolution occurrences; and identified evolution patterns. We have described how those information is modeled and stored in the EPOSDB. The Project Characterizer Tool and Evolution Registrar Tool are implemented to record project and evolution data to the experience database. We have also presented how to implement the approaches to manage enactable and enacting process model (i.e. task network) in EPOS. They manipulate the task network
### Figure 8.14: Evolution Category

<table>
<thead>
<tr>
<th>Where</th>
<th>Why</th>
</tr>
</thead>
<tbody>
<tr>
<td>Category</td>
<td>Functional</td>
</tr>
<tr>
<td>Source</td>
<td>Errors and mistakes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>When</th>
<th>By whom</th>
</tr>
</thead>
<tbody>
<tr>
<td>Discovery</td>
<td>Implementation</td>
</tr>
<tr>
<td>Correction</td>
<td>Implementation</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>How</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Category</td>
<td>Tuning and correction</td>
</tr>
<tr>
<td>Source</td>
<td>Rework</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Category</th>
<th>Internally</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source</td>
<td>Project member</td>
</tr>
</tbody>
</table>

### Figure 8.15: Evolution Cost

<table>
<thead>
<tr>
<th>Cost</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Estimated</td>
<td>Estimated</td>
</tr>
<tr>
<td>Actual</td>
<td>Actual</td>
</tr>
<tr>
<td>Difference</td>
<td>Difference</td>
</tr>
<tr>
<td>Percentage</td>
<td>Percentage</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Calculate</th>
<th>Calculate</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
8.9. Chapter Summary

<table>
<thead>
<tr>
<th>Overview</th>
<th>General</th>
<th>Categorization</th>
<th>Cost</th>
<th>Description</th>
</tr>
</thead>
</table>

**Description of evolution:**

- The error was discovered by the quality control team.
- If ‘Remove’ is selected on the product specification screen, the system crashes.

**Changes and correction actions:**

- Jarn Nygjerd corrected the problem by adding a simple test before trying to remove the product.

Figure 8.16: Evolution Description

layout, schedule and resource before and during enactment. The two basic operations to manipulate the task network layout are task insertion and removal. They are described and illustrated with screen pictures in EPOS. We also demonstrate how the task schedule and resource can be revised. The empirical evolution planning approach is supported by using the Project Selector Tool and Evolution Analyzer Tool against the EPOS experience database. Finally, we present two student works which are supervised by the author. Both student projects adopt the evolution categorization framework to classify and record project and evolution data to better plan new similar projects. One of those projects has implemented an experience database for the software company against which we are conducting a case study. Another project has implemented an experience database prototype on World Wide Web. It is also worthwhile to mention that the tools presented in this chapter are actively used to collect the case study information, and then retrieve the analytical results which are reported in the case study in chapter 9.
Chapter 9

Case study

9.1 Introduction

In this chapter, we present a case study conducting against a real software company. In this case study, we apply our categorization framework (in chapter 6) to classify and collect process evolution occurrences during selected projects. Further, our proposed approach (in chapter 7) is adopted to provide empirical baselines for effort distribution over project life time, error rate, error removal effort and evolution profile. The company’s ability in making estimates and the extent of budget overruns are also determined during the case study. Collected evolution data are then analyzed and typical patterns of evolution are then identified. To improve the company’s ability to anticipate evolution, we also identified some empirical relations between external project characteristics on one hand and process evolution and estimation capability on the other hand. Most frequent and typical evolution patterns are recognized. The realistic, collected evolution data from the case study are packaged in EPOS experience database. Parts of process models from the studied company are modeled in EPOS. The corrective reaction of identified evolution patterns are then realized by invoking manipulating operations to evolve task network in EPOS (chapter 8). Based on gained experiences and findings, improvements are suggested to both the software company and EPOS.

The main business stream of the studied company is provision of financial and banking applications, covering a considerable Norwegian marked share. The company’s strategic goal is to continuously improve its competitive advantage by delivering product on time, within budget, of high quality, and thus increase customer satisfaction. The birth of the case study stems not only from a close cooperation with our Software Engineering Group, but also the recognition of persistent late project delivery. Established baselines which represent current state of projects at the studied company, are useful in making effective strategic decisions, as well as in providing quantitative evidences on improvement effort. Due to the confidentiality of the data reported, the studied company will be referred to as XXX.

The chapter is organized as follows. Section 9.2 presents the experimental paradigm of Basili which serves as a guideline in conducting our case study. Section 9.3 gives an overall context of the studied software company, including its general background information, its quality system and our assessment exercise on current practices at XXX. Section 9.4 presents the case study design, specifying the objectives, strategies, research questions and planned activities to be carried out during the case study. Section 9.5 reports completed activities with their deliverables. Section 9.6 presents empirical results followed by a thorough analysis and discussion of collected evolution data in section 9.7. In this analysis section, we also identify most typical evolution patterns and their empirical relations to external project characteristics. Section 9.8 illustrates how such typical evolution patterns are man-
aged by invoking task network manipulating operations in EPOS. Improvements are then suggested both to the company and EPOS in section 9.9, before we discuss gained benefits, research questions, learned lessons and threats to validity of the case study in section 9.10.

9.2 The Experimental Paradigm

Our case study attempts to conform to a theoretical approach, called the *Experimental Paradigm* by Victor Basili [Bas92]. This paradigm consists of four major activities:

1. **Hypothesis formulation**: This activity will mainly formulate the hypotheses based on the theoretical framework and the objectives of the experiment. The stated hypotheses are used to determine the data that must be collected providing quantitative evidences. The three following activities will thus evaluate the hypotheses and their implications. Since we do not have much control over actual behavior of selected projects in the case study, we prefer to state a set of research questions instead (see section 9.4.2).

2. **Data collection**: This gathering activity involves conducting a series of interviews with relevant actors or studying the existing process. A set of questionnaires or forms will be elaborated and installed to collect wanted data with least cost as possible (see section 9.5).

3. **Data analysis**: The gathered data will be analyzed and evaluated within the context of the theoretical framework. The stated research questions should subsequently either be validated or refuted (see sections 9.7 and 9.10).

4. **Improvement**: The lessons gained and learned from the data analysis activity will constitute a quantitative foundation for proposing improvement opportunities. The existing baselines will hence be modified reflecting the new insights and understanding after conducting the experiment (see section 9.10).

9.3 Case Study Context

9.3.1 Background

XXX is a Norwegian software provider and computer center for banks and financial institutions with 329 employees of which 89 are working with software production. It is located in three different sites, with the biggest development department in YYY-city. The proposed case study will be carried out closely with a System Development Division (hereafter referred to as SDD) in YYY. Besides SDD, there exist Product Division, Maintenance Division, Sale Division and Service Center. XXX maintains 139 different software products. Its mainframe is running ORACLE on VAX/VMS and DB2 on IMB/MVS. Most of the applications developed at XXX, are running distributed on IBM-compatible PCs under MS-DOS/Windows, and local network against UNIX- or NT-based communication servers. XXX distinguishes between commission and project depending on required work amount. A typical duration for project is ranging from 0.5 to 1.5 year, with totally 5000-8000 hours. A typical commission which mainly fixes minor changes or enhancements in existing system, has a budget of 250-300 hours.
XXX is ISO-9001 certified since January 1995 and thus has defined processes for development and maintenance software system. These processes are documented in a Quality System (see below for details). XXX has been collecting a marginal set of measures reflecting experiences from completed projects for 4 years. Product and process measures are collected regularly with filled-out paper forms or questionnaires. Typical measures being collected are:

- Budgeted and accumulated actual performance of projects, i.e. spent effort hours and elapsed time are monthly reported.
- Deliverability for projects/commissions, i.e. the percentage of projects or commission are delivered on time and within budget.
- The number of post-release errors (i.e. after initial installation) per product reported by customer to Service Center, and their required correction effort.
- Annual survey of customer satisfaction with both products and services.
- Annual survey of personnel satisfaction with work situation, upper management, social relation between colleagues.

SDD suffers from late project delivery and realizes that constant evolution during project is a major cause. Annual figures in 1995 indicate that only 82% of commissions and 26% of projects are delivered on time and within budget. In addition, 6.3 post-release errors per product were reported, and average 7 hours were spent to correct each of them. Those error figures do not seem as disastrous to many software companies, but SDD however finds them unacceptable with respect to its required availability for its online systems.

SDD has realized that it is important to learn from past experience on process evolution. To do so, SDD prevents new projects from going into the same pitfall, exceeding planned budget, and at the worst case ending up as fiasco. The SDD has therefore in that context stated as improvement goal to:

- thoroughly record and analyze causes of process changes and their corrective actions during development projects. In that context, it also wants to establish baselines reflecting the extent to which evolution has affected the outcomes in completed projects.
- improve accuracy of project estimates on effort required for the entire project and for each phase in development life cycle.
- reduce the number of post-release errors and their removal effort. That is, most errors should preferably be detected and removed before delivery. Currently, XXX does not collect any data on pre-release errors. We therefore need to establish an error baseline before stating any quantitative improvement goal.

### 9.3.2 Quality System of XXX

The Quality System (QS) of XXX relies on the recognition that desired quality is not achieved by coincidence, but by systematically following defined processes. The QS describes strategies, activities, organization and management systems of XXX by answering questions like: what to do; who is responsible; how to carry out; and how to demonstrate that work is done. The XXX's strategy describes
the vision, objectives and quality policy prevailing at any time. XXX's activities are divided into defined activity areas encompassing activities which logically belongs to each other. Defined activity areas cover aspects such as Management, Development, Installation, Maintenance, Support, Sale and Purchase. XXX is organized in divisions responsible for specific competence domain in one or several activity areas. Projects are usually staffed by personnel from different divisions. The QS can be considered as a collection of process models templates or standards in XXX and is organized hierarchically by four levels:

1. **Quality Handbook** describing overall requirement to the quality system.

2. **Procedure Handbook** describing prescribed course of action for each defined activity area.

3. **Handbook** specifying in details each procedure defined in level 2 when necessary.

4. **Form/Questionnaire** consisting necessary forms used in each procedure, e.g. progress report, test log, review report, ....

In the case study, there are only two handbooks which are of relevance and interest for our purposes. They are thus described below. The first one expresses the project model and belongs to the Procedure Handbook for Support activity area. The second one describes the system development model and is an integral part of the Procedure Handbook for Development activity area.

![Figure 9.1: Project Model of XXX](image)

Figure 9.1 depicts the project model of XXX and consists of five major phases.
9.3. Case Study Context

**Initiate** Project is supposed to deliver an unique solution and thus requires specialized tasks. Those tasks may be completely distinct from trivial ones which are performed in daily basis. This activity will thus evaluate incoming customer request and then determine whether it can be run within a context of a project. Several criteria of the customer request must be taken into consideration during evaluation such as complexity, number of participants, cost, vulnerability, size, duration, ….

**Make decision** After the decision on project initiation is approved, a project mandate is produced by this activity. A project mandate usually embodies: i) project organization, i.e. project participants, their roles and responsibilities; ii) project constraints, including planned cost, budget and duration; and iii) project description, containing expected results and benefits.

**Planning** This activity results in a project plan which describes in details, apart from the content in project mandate, concrete activities, phases, and milestones. In this project plan, activities in the next coming phase are fully elaborated, while the activities in later phases are described coarsely.

**Realize** This activity includes phases which are similar to those in waterfall life cycle. Those phases are defined in XXX's system development model which is depicted in figure 9.2. Except from trivially technical documents such as, Requirement Specification, Design Specification, Test plan, Source Code, …, this activity must monthly generate reports on project status and progress. A milestone plan with detailed activity description is produced at beginning of each phase.

**Close up** When finishing the realization phase, the developed system is delivered to Product Division who is responsible for installation and user training. The project is declared to be concluded, and a final report is then produced to report the actual status on cost and duration of the project. An experience report is also generated and archived for later use.

9.3.3 Assessment of Current Practices at XXX

We have in 1996 participated in a national pre-project, called Software Process Improvement for better Quality (SPIQ) [Con96]. SPIQ is planned to be executed as a later user-driven project in 1997-2001 by four research institutions and 10-12 software organizations. The project will be partly funded by the Norwegian Research Council and by the involved industrial partners which are mainly small and medium enterprises. They operate in different domains, apply different development technologies, and represent different sizes and company culture. The process maturity and awareness vary greatly among the industrial participants. That is, the improvement approach applied must be evaluated and adapted carefully. During the first half year of the pre-project, we focus on local assessment of current practices to obtain a baseline, and identify problems and improvement areas. A common improvement framework will be consolidated based on well-known SPI-methods, to suite conditions in Norway. The partner assessments lay a foundation for making concrete improvement plan. In the later user-driven SPIQ project, we plan to recruit more industrial participants, and actually to implement defined improvement plans. The final deliverable of SPIQ project is to consolidate experiences and results in form of a Method Handbook for Norwegian Industry.

XXX is one of the industrial partners in SPIQ. The author has therefore assisted XXX to carry out an assessment of its current process status. A specialized questionnaire and interviews of relevant actors have been used. This assessment was carried out between June and August in 1996. Two development
projects have been selected for the assessment. Five persons from SDD (a senior executive, a quality/security manager, a process manager, two project mangers, and two developers) have filled out the questionnaire and been interviewed. The assessment has concluded with following observations and findings:

- Process standards and procedures (i.e. generic process models) are well defined and documented in the Quality System at company level. They are then tailored for each project by a project manager to produce an adequate project plan (i.e. enactable process model). Such tailoring process is heuristic and experimental, in the sense that a large number of assumptions and guess work are embedded in the plan. Project estimates in the project plan are thus largely influenced by the knowledge, skills and the unconscious experiences of the project manager.

- The collected process and product measures from completed projects have been recorded in paper-based forms which are poorly structured and thus seldomly used in planning and estimating of new projects. Project estimates are elaborated by “thumb-rules”, rather than by applying defined estimation methods in the XXX’s handbook. This is mainly due to the lack of easily accessible, empirical data on which those estimation methods essentially depend.

- There exists no visible mechanism (e.g. check lists, verification, ...) to indicate the degree in which project personnel follows and conforms to defined process description (no verification of process conformance). Unrealistic deadlines, high workload and lack of necessary resources are major causes for ignoring or neglecting vital process steps.

- Many useful and common product metrics, such as size, complexity, pre-release error data, ... are not collected. It is therefore difficult to provide significant evidence on possible improvement
of product quality. The only collected product measure is the number of post-release errors (i.e. after initial installation) per product. The lack of relevant product measures makes it hard to recognize quantitative relations between product properties and development process.

- Evolution experiences from completed projects are documented in Experience Report, but unfortunately drowned with other irrelevant information. Such experiences are recorded in an unstructured and unsystematic manner. Therefore, valuable experiences from prior projects are difficult to access, analyze, and reuse to support decision making process in future projects. Everybody at XXX has an intuitive, rather than quantitative, understanding of evolution during projects and of their potential impact on project outcomes.

- Many changes during project execution originate from customers who constantly revise initial requirements due to e.g. misunderstanding, omission and ambiguity. Such changes often result in unpleasant delays and even re-scheduling of the initial project plan. Despite the severity of such changes, there exists however no quantitative indication on their frequency and impact. Projects which are contracted with fixed price, has often to bear the additional cost.

Based on above findings, we have identified following potential improvement initiatives. They have also laid foundation for identifying the objectives for our case study.

- Structure existing project information to facilitate easy access, and thus effective reuse. Further collected data should be analyzed to better understand and anticipate process evolution frequency and their impacts during development projects.

- Improve accuracy of project estimates on effort and duration by actively using historical data from prior projects.

- Define and collect product measures. This must be carried out according to which product aspects need to be improved.

### 9.4 Case Study Design

In this section, we first present overall objectives of the case study, followed by an outline of various strategies for satisfying those objectives. Secondly, we state a set of pertinent research questions to be investigated by the case study. Finally, we describe planned activities which are an elaboration of the identified strategies.

#### 9.4.1 Overall Objectives and Strategies

In light of the improvement goals of XXX and findings from the assessment exercise above, we have defined a set of objectives for our case study.

**01. Getting baselines** The case study aims to establish company-wide baselines for XXX with respect to estimate accuracy, effort distribution, pre-release error profile and evolution. As an experiment, a new technology is applied. In this case, we apply our categorization framework to collect and classify process evolution occurrences. The collected data will be packaged into an experience database.
O2. Assessing baselines  The obtained baselines are then used to plan and make estimates on effort and duration of new pilot projects. Such a controlled experiment is carried out with a hypothesis which states that process evolution is better anticipated and the estimates are thus more accurate than the existing baselines. In addition, the collected evolution data are used to improve the ability to anticipate changes and predict their impacts. The gained results from the pilot projects are evaluated. On the basis of that evaluation, XXX’s process models are revised to incorporate the validated technology.

O3. Validating EPOS  The case study will provide realistic data on most frequent evolution and their corrective actions. Such actions found basis for validating EPOS’s approaches in managing process evolution. That is, the ability to dynamically manipulate task network layout, schedule and resources must be validated by the collected evolution occurrences.

Due to limited amount of resources at XXX to assist the performance of the case study, most activities are done either by the author or by supervising student works. However, we are granted access to the paper-based project archive, containing all information about both completed and ongoing projects. In the following, we point out strategies or means for achieving the objectives above.

Project sample  The studied projects should be representative for ongoing and future projects. That is, both typical completed and up-starting projects should be selected. Such a selection process requires assistance from experienced personnel at XXX.

Paper work  The selected, completed projects from the project archive are examined. Such a task is expected to be time-consuming, and thus needs to be performed in a systematic manner.

Interview  Intensive interviews or informal chats with project managers and project participants to clarify ambiguous or missing information from the project archive. E.g. a particular change needs to be elaborated.

Collection  To determine existing estimate accuracy of XXX, we have to collect and compare the initial project estimates against the actual performance values. A quantitative baseline for pre-release error must include the number of errors which are found and corrected and their removal effort. Each error must be associated with a set of characteristics such as type, origin, location, causes, costs and impacts.

Analysis  The collected data are useless without performing appropriate analysis. That is, collected data must be analyzed to demonstrate relevant aspects such as estimation capability, relation between process evolution and external project characteristics, and etc .... It is also relevant to see the process evolution causes and their associated impacts. Such analytical data is useful to validate EPOS’s ability to manage process evolution.

Commitment  It is crucial to obtain commitment and trust from both senior management, project managers and project members to achieve fruitful and valid results. The management at all levels and all personnel involved must be correctly informed about the objectives of the case study. By doing so, we prevent from collecting only fictitious data. E.g. the Hawthorne effect: people work better merely because of process change or being observed.
9.4.2 Research Question for the Case Study

By conducting the case study, we want to have answers for following research questions:

Q1: There are superior number of process evolution which imply changes to the project plan (i.e. enactable and enacting process model), rather then to the XXX’s Quality System (i.e. generic process model). That is, unplanned process evolution is more frequent and thus deserves more attention than planned one.

Q2: The application of the categorization framework in the case study sufficiently captures and categorized most process evolution occurring during the studied projects.

Q3: There exist empirical relations between external project characteristics and identified evolution patterns. Such relations are useful to anticipate and predict changes for “similar”, new projects. That is, the estimates for new projects become more precise

Q4: The proposed empirical evolution planning approach describes the necessary course of actions to establish and use baselines from previous project experiences.

Q5: Operations manipulating task network layout, schedule and resources in EPOS (see chapter 8.4) are adequate to realize corrective actions of identified evolution patterns from the case study.

9.4.3 Planned Activities for the Case Study

In this section, we elaborate the strategies above to a set of activities which are carried out in the case study. Those activities are divided into six groups.

A1. Preparation This comprises development of necessary questionnaires or forms used in the case study. Completed and up-starting projects will be selected for the case study.

A2. Data Collection This embodies activities to collect relevant measures from completed projects which are selected by A1, by looking in the project archive.

A3. Monitoring New forms (from A1) are introduced to personnel and incorporated to the project plan of the selected, up-starting project. They are also trained in using the new forms to record process evolution. Activities A2 and A3 together will establish baselines for XXX.

A4. Tool provision This encompasses collaborative activities between XXX, students and the author to establish an experience database at XXX and in EPOS. A set of associated tools are also implemented to collect and analyze evolution data.

A5. Reuse of Experience Data New pilot projects are selected to exploit assembled experience data in order to improve the planning and execution. The existing baselines which are established by A2 and A3, are revised to account for any detected deviation.

A6. EPOS Validation Collected evolution data are exported to and packaged in EPOS experience database for validating approaches which are implemented to deal with process evolution.

Figure 9.3 depicts the phasing of six activity groups which are treated in the following subsections.
9.4.3.1 A1 - Preparation

This preparation activity includes three sub-activities: i) determining the data to be collected during the studied projects; ii) elaborating appropriate forms used to collect necessary measures; and iii) selecting completed and up-starting projects for the case study. Following is detailed description of all three activities:

A1.1 - Measure Determination

The objectives and strategies mentioned above serve as an input to this activity whose intention is to determine what to be measured and collected.

A1.2 - Form Elaboration

Relying on the results from A1.1, we proceed to define necessary paper-based forms. Elaborated forms must satisfy following criteria: completeness, understandability and well-structure. It is therefore important to use terminologies which are commonly used at XXX. The new forms are intentionally adopted in activity A3 with the up-starting project. However, they can also be used to make the data collection process from completed projects (activity A2) more systematic and efficient.

A1.3 - Project Selection

The validity of the obtained baselines are strongly influenced by the selection of projects for the case study sample. We have therefore identified a set of substantial criteria for such selection.

Representative The selected projects must be representative for XXX in order to assure relevancy, validity and replicateability of the achieved results. That is, the gained baselines must facilitate
9.4. Case Study Design

future use and projection.

Well-documented The selected projects must demonstrate a certain extent of detail and completeness of their reports in the project archive. This is a vital prerequisite to obtain required correctness of the overall picture.

Recent However, we can not guarantee to find whatever we need in the project archive about a particular project. It is thus necessary that the involved personnel must be available for questions and interviews. To be able to get as correct information as possible, the selected projects should be recent in time, and personnel still recall most of the incidents.

Varied The selected projects should represent a broad spectrum of characteristics with respect to e.g. project type, customer type, sub-contractor type, duration, number of personnel, application type, .... This criterion is crucial to recognize empirical relations between external project characteristics on one hand and process evolution and estimation capability on the other hand.

Selection of the up-starting projects for the case study also relies on the same criteria above. Moreover, we need to select a up-starting project whose personnel are motivated and fully committed to the case study.

9.4.3.2 A2 - Data Collection

This data collection activity comprises of two sub-activities: data archaeology; and presentation and storing of collected data.

A2.1 - Data Archaeology

Input to this activity is a huge pile of reports from selected, completed projects. It is first of all necessary to get a clear insight into the structure of the different reports, i.e. what kind of information is expected to be found in each report type. The new forms which are developed in A1, are then used to collect relevant data. Due to the fact that the new forms introduce new measures to be collected, it is possible that required information may not be retrieved neither from the report nor from the involved personnel.

A2.2 - Data Presentation and Storage

The collected data should be presented in an easily understandable manner, i.e. in tabular or diagrammatic forms. The collected figures are then presented to the involved personnel for verification and then to upper management taking necessary actions. Experience databases which are developed in activity A4, are used to store such collected project data facilitating for easy access, comparison and analysis.

9.4.3.3 A3 - Monitoring

This activity aims at applying new forms to collect data which have not been collected previously, in the up-starting project. The intention is to evaluate the usefulness as well as the applicability of those forms in a real project. Activity A3 includes of three sub-activities below:

A3.1 - Introduction
Project personnel are informed about the motivation and objectives of the case study at one of the project start-up meetings. The categorization framework for process evolution is also presented. It is crucial to communicate with their common terminologies, and perceive thing from their viewpoints. Otherwise, the case study has a marginal chance to survive and succeed. Introduction material, including the new forms, is presented and distributed at the meeting.

A3.2 - Participation and Collection

The author actually becomes a project member of the studied up-starting project, and thus frequently participate the weekly status meeting. At those meetings, the author assists project personnel to categorize and record process changes by filling out the form. The purpose of participation at the meeting is to capture process evolution occurrences which are otherwise not perceived by the project personnel. Besides, the author also assists the project manager to fill out the form which characterizes the project in study. The collected forms are carefully reviewed. Any omission or ambiguity must be clarified with the involved persons. Accumulated results and findings should also be reported at the weekly meeting.

A3.3 - Verification

When the studied, on-going project is completed, the collected data are presented to the project group for verification. Eventual conflicts or discrepancies are then clarified before the results are incorporated with those obtained from activity A2. The final results of this activity is a set of empirical baselines which represent the current status of the company. We hereafter refer to them as company-wide baselines.

9.4.3.4 A4 - Tool Provision

The key objective of this activity is to provide tool support to collect, record, store, and analyze the project data. This activity is divided into four sub-activities below:

A4.1 - Experience Database Prototype for XXX

A group of six students is assigned to implement an experience database prototype and its associated supporting tools for XXX. This database will at first structure and record exiting metric data at XXX. In addition, it provides support for presenting, analyzing, and generating reports which have otherwise produced manually.

A4.2 - Simple Evolution Registration Tool for Activity A2

The project personnel in the up-starting project complain about the great amount of effort which is spent to fill out the new paper-based forms. Due to the absence of appropriate tool support for recording evolution data, we recognize a need for having a simple computer-assisted registration tool. Such tool will simply visualize the new form, store collected data in a database, and ease the registration process considerably.

A4.3 - Integrated Experience Database for XXX

When both activities A4.1 and A4.2 are completed, it is necessary to integrate them into a single and integrated environment for XXX. This upgraded environment (i.e. experience database) is expected to administrate both existing and new measures which are introduced in the case study.
9.4. Case Study Design

9.4.4 - Experience database in EPOS

This activity focuses on extending EPOS to store various project experiences in the database. An experience database and associated tools for recording and analyzing process evolution data are planned to be integrated into EPOS for further experiment. The new approaches described in chapter 7 are also realized in EPOS to better manage process evolution on the basis of the collected data.

9.4.3.5 A5 - Reuse of Experience Data

The obtained baselines are useless without thoroughly applying to planning in new projects, and then demonstrating evidences on improvement of predictability. This activity which can be regarded as a controlled experiment, is divided into three sub-activities.

A5.1 - Retrieving/Predicting

New pilot projects are selected for the experiment. The project managers use the existing methods for making project estimates and schedules without any knowledge of empirical data from the case study. The obtained baselines are then presented to the project managers who are encouraged to revise his/her initial estimates to account for the new insight. The most similar completed project is first selected and used as baseline project. Experience from the baseline project or from all completed projects found a quantitative basis for making new estimates. Such new estimates therefore anticipate the most likely process evolution patterns.

A5.2 - Tracking

The new pilot projects are tracked, and process evolution occurrences are recorded as we did with the up-starting project earlier (activity A3). When the pilot projects however confront with an unanticipated process evolution, the integrated experience database (from activity A4.3) is then used to access information on corrective actions and their actual impacts from similar situation. By doing so, useful experiences from completed projects are used to guide further actions.

A5.3 - Packaging/Refining

At completion of the pilot projects, newly gained experiences are packaged into the integrated experience database, and existing baselines are revised accordingly.

9.4.3.6 A6 - EPOS Validation

The intention of this activity is to validate EPOS approaches to manage process evolution which are described in chapter 7.5. The collected evolution data at XXX are exported and recorded in EPOS experience database to analyze and recognize evolution patterns. The collected evolution data from real projects also describe the actions which are actually taken to correct such situations. The project model and system development model of XXX are first described and expressed in EPOS formalism, SPELL. An example of a project plan is then constructed as a task network within EPOS context. EPOS operations which are used to manipulate task network layout, schedule and resources are then invoked to “simulate” the corrective actions associated to recognized evolution patterns.
9.5 Case Study Performance

In this section, we present the performance of planned activities in section 9.4.3. Along with the presentation of completed activities, concrete results or deliverables are also mentioned.

9.5.1 A1 - Preparation

This activity has been completed and resulted in four paper-based forms to characterize external project characteristic; record process evolution; summarize project outcomes; and categorize error. In the following, we describe the process in which those forms are elaborated.

Project Characteristic Form

One of research questions stated in 9.4.2 wants to demonstrate empirical relations between project characteristics and process evolution. To do so, we need to identify a set of project measures which possibly influence the frequency and type of evolution occurrences. A discussion on such project profile measures/indicators is described in section 6.4, and founds a basis to determine relevant project measures. This new form is used to specify project and product characteristics which are retrieved at project start. Such characteristic can be categorized into two groups: profile or estimate.

Project profile includes following objective measures:

- Name of project manager. We have mentioned above that project managers' skill and knowledge strongly influence the project outcome.
- The number of both internal and external project participants. The ability to cooperate and communicate can be influenced by the staff size or different background and culture.
- The development method which is selected for the project, and tool supports which are used for project performance and tracking.
- Project type (i.e. development, upgrading, maintenance). We believe that project types represent a variety of skill, knowledge and competence of the staff about the problem domain.
- Name and type of customer and eventual sub-contractor or supplier.
- Project constraints which are negotiated in the contract. They represent project budget (i.e. the project price is either fixed, or negotiate-able when necessary, or fully funded by the customer, or paid by the company); delivery date (i.e. customer and the company either agree on a fixed delivery date, or negotiate during project execution); and requirement stability (i.e. user requirements are agreed to be either frozen or continually revisable).
- Risk concerns the availability of competence required by the project, the newness of technology adopted in the project, and subjective judgment on the project difficulty. The chosen values for each measure are self-explainable.

Project estimates represent the planned budget which is expressed in term of money and working hours. The estimated effort distribution over development phases and period (i.e. months in our case) is also specified together with the project duration. The planned dates to complete particular deliverables during development life cycle are also estimated.
Product profile concerns with the language and platform of both the server and client application. The communication protocol is also of relevancy in bank application. The selected languages and platforms represent the most common ones in both previous, ongoing and possibly future projects. We believe that such product measures have major impact on project behavior and outcome.

Product estimates represent the common product characteristics such as: developed, generated or reused line of codes; generated number of user-interface screens; function point; number of produced rapport; number of database table; number of incoming and outgoing transactions.

Figure 9.4 depicts the paper-based form for characterizing project property for XXX.

Evolution Registration Form

This form is used during project execution to record process evolution occurrences. Such collected evolution occurrences are categorized based on our framework which is presented in chapter 6. The framework provides three levels for classifying a process evolution. In particular, a process evolution is classified by six dimensions, their aspects and categories. The aspects and categories must be specialized to accommodate to the company's particular situation. This specialization has been done and presented as an example in chapter 6. In addition, the Evolution Registration Form also collect the estimated and actual cost in correcting the recorded process evolution. Figure 9.5 depicts the paper-based forms for recording process evolution at XXX. Note that the defined dimensions, aspects and categories are presented as three vertical columns in the main table of the form.

Final Project Status

One of the case study objectives is to determine the accuracy of estimates. We have collected project and product estimates in the Project Characterization Form (depicted in figure 9.4). This form will thus be used to collect actual project performance as well as product measures. It means that the same measures of project and product estimates in Project Characterization Form, are included in this form and represent actual project and product status respectively. Apart from those measures, we also collect the actual efforts which are spent in removing error, reviewing document and inspecting code. The actual numbers of error found and corrected in different test phases are also recorded. Each error is recorded and categorized by using the Error Report which are presented below. The total frequency of process evolution occurrences and their correction cost are also included in this form. Figure 9.6 depicts such a paper-based form.

Error Report

The case study objectives also require to establish a baseline of error profile for XXX. The existing forms for collecting errors during test phases at XXX only include a textual description of error and corrective action associated to a document or a code module. XXX does not have any error categorization, and thus no perception of error types and impacts. We have suggested a revision of existing form to incorporate error categories and their actual removal effort. There are several error categorization systems in literature, but we have limited the number of error categories based on our particular
## PROJECT CHARACTERIZATION FORM

<table>
<thead>
<tr>
<th>Project ID:</th>
<th>Project name:</th>
<th>Filled out by:</th>
<th>Date:</th>
<th>Numb of page:</th>
</tr>
</thead>
</table>

### Project profile:

<table>
<thead>
<tr>
<th>Project manager</th>
<th>Name:</th>
<th>Employee ID:</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Number of project member</th>
<th>Internal:</th>
<th>External:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Other:</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Applied development model/method</th>
<th>Waterfall</th>
<th>Prototyping</th>
<th>Spiral</th>
<th>Iterative</th>
<th>Incremental</th>
</tr>
</thead>
<tbody>
<tr>
<td>Own model</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Other:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tool support to project management</th>
<th>MS-Project</th>
<th>SuperProject</th>
<th>MS-Office</th>
<th>WordPerfect</th>
<th>Other:</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Project type</th>
<th>Development</th>
<th>Upgrading</th>
<th>Maintenance</th>
<th>Other:</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Customer type</th>
<th>Bank</th>
<th>Internal</th>
<th>External</th>
<th>Joint development</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Customer name</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Sub-contractor/Supplier</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Contractual constraints</th>
<th>Price</th>
<th>Negotiate-able</th>
<th>Fully funded</th>
<th>Own risk</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Fixed</td>
<td>Negotiate-able</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Req. stability</th>
<th>Frozen</th>
<th>Revisable</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Risk</th>
<th>Necessary competence</th>
<th>Non-existent</th>
<th>External existent</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Partly available with training</td>
<td>Internal available</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Adopted technology</th>
<th>New</th>
<th>Internal unknown</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Partly known</td>
<td>Well-known</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Difficulty</th>
<th>Minor</th>
<th>Middle</th>
<th>Manageable</th>
<th>High</th>
</tr>
</thead>
</table>

### Project estimates:

<table>
<thead>
<tr>
<th>Planned budget</th>
<th>in money:</th>
<th>in working hour:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Effort distribution (%)</th>
<th>on development phases</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Planning:</td>
</tr>
<tr>
<td></td>
<td>Analysis:</td>
</tr>
<tr>
<td></td>
<td>Integration test:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Pilot:</th>
<th>Documentation:</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Other:</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Planned scheduled</th>
<th>From:</th>
<th>To:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Progress (based on milestones)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plan document:</td>
</tr>
<tr>
<td>Design specification:</td>
</tr>
<tr>
<td>Integration test:</td>
</tr>
<tr>
<td>Acceptance test:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Applied estimation method</th>
<th>Break-down</th>
<th>Outer product characteristic</th>
<th>Other:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
9.5. Case Study Performance

Product profile:

<table>
<thead>
<tr>
<th>Product ID:</th>
<th>Product name:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server</td>
<td></td>
</tr>
<tr>
<td>Platform</td>
<td>MS-Windows □</td>
</tr>
<tr>
<td></td>
<td>Windows-NT □</td>
</tr>
<tr>
<td>Language</td>
<td>C □</td>
</tr>
<tr>
<td></td>
<td>Progress □</td>
</tr>
<tr>
<td>Client</td>
<td></td>
</tr>
<tr>
<td>Platform</td>
<td>MS-Windows □</td>
</tr>
<tr>
<td></td>
<td>Windows-NT □</td>
</tr>
<tr>
<td>Language</td>
<td>C □</td>
</tr>
<tr>
<td></td>
<td>Progress □</td>
</tr>
<tr>
<td>Communication protocol</td>
<td>X.25 □</td>
</tr>
<tr>
<td></td>
<td>Other: □</td>
</tr>
</tbody>
</table>

Product estimate:

| Developed LOC | 0 1000 □ | 0 5000 □ | 0 10.000 □ | 0 50.000 □ | 0 100.000 □ | Flere □ |
| Generated LOC | 0 1000 □ | 0 5000 □ | 0 10.000 □ | 0 50.000 □ | 0 100.000 □ | Flere □ |
| Reused LOC    | 0 1000 □ | 0 5000 □ | 0 10.000 □ | 0 50.000 □ | 0 100.000 □ | Flere □ |
| Generated # of screen | 0 20 □ | 0 60 □ | 0 100 □ | Flere □ |

Function Point

| Number of produced report |
| Number of database table  |
| # of incoming transaction-type |
| # of outgoing transaction-type |
| # of internal transaction-UI  |
| # of external transaction-UI   |

Product description:

Figure 9.4: Project Characterization Form
<table>
<thead>
<tr>
<th>Project ID:</th>
<th>Project name:</th>
<th>Filled in by:</th>
<th>Date:</th>
<th>Evolution ID:</th>
</tr>
</thead>
</table>

### WHERE source of change
- **Project internal**
  - Executive
  - Quality manager
  - Steering committee
  - Project manager
  - Project member
- **Project external**
  - Customer
  - Sub-contractor/Supplier
- **Other**
  - Competitor
  - Market trend
  - Technology

### WHY reason of change
- **Technical/Functional**
  - Misunderstanding
  - Ambiguity
  - Omission
  - Error
  - Revised requirement
  - Better insight
  - Lack of competence
- **Organisational/Managerial**
  - Delay
  - Improvement
  - Postponement
  - Lack of res.
  - Under-estimate
  - Over-estimate
  - Re-prioritising

### WHAT affected entity
- **Technical document/Deliverables**
  - Requirement
  - Design
  - Source code
  - Test plan
  - Deliverable
  - User
  - Installation
  - Operation manual
- **Managerial document**
  - Project mandate
  - Project plan
  - Milestone plan
  - Project handbook
  - System development handbook
- **Tool support / Software**
  - Development
  - Project management
  - Purchased SW
- **Managerial**
  - Resource distribution
  - Schedule/Progress
  - Budget

### WHEN
- **Detection time**
  - Planning
  - Requirement
  - Design
  - Realisation
  - Acceptance-test
  - Pilot
- **Correction time**
  - Planning
  - Requirement
  - Design
  - Realisation
  - Acceptance-test
  - Pilot

### HOW corrective action initiated
- **Adjustment/correction (fire extinguisher)**
  - Rework
  - Prolong ongoing activity
  - Postpone
  - Split
  - Merge forthcoming activity
  - Revising/re-estimating
  - Release resource
  - Obtain extra resource
- **Innovative (long-term investment)**
  - Infuse new development technology
  - Infuse new project management technology
  - Training

### BY WHOM corrective act. performed
- **Project internal**
  - Executive
  - Quality manager
  - Steering committee
  - Project manager
  - Project member
- **Project external**
  - Customer
  - Sub-contractor/Supplier

### Description of change:

### Change impact on project outcome:

### Description:

**Estimated correction hour:**

**Estimated cost:**

**Actual correction hour:**

**Actual cost:**

### Description of corrective action:

---

Figure 9.5: Evolution Registration Form
### Final Project Status Form

**Project ID:**

**Project name:**

**Filled out by:**

**Date:**

#### Actual Project Status:

<table>
<thead>
<tr>
<th>Actual cost</th>
<th>in money:</th>
<th>in working hour:</th>
<th># versions of project plan:</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Estimate deviation (+/-)</th>
<th>in money:</th>
<th>Deviation (+/- %):</th>
<th>in working hour:</th>
<th>Deviation (+/- %):</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Actual duration</th>
<th>from:</th>
<th>to:</th>
<th>Deviation (+/- # of day):</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Effort distribution (%)</th>
<th>Planning:</th>
<th>Project administration:</th>
<th>Documentation:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analysis:</td>
<td>Design:</td>
<td>Realisation:</td>
<td></td>
</tr>
<tr>
<td>Integration test:</td>
<td>System test:</td>
<td>Acceptance test:</td>
<td></td>
</tr>
<tr>
<td>Pilot:</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Productivity</th>
<th>Product size/Actual cost:</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Error removal effort (# working hour)</th>
<th>Integration test:</th>
<th>System test:</th>
<th>Acceptance test:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maintenance:</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>QA-effort (# working hour)</th>
<th>Structured walk-through/review of document:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code reading:</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Personnel satisfaction</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Customer satisfaction</th>
</tr>
</thead>
</table>

#### Actual Product Status:

<table>
<thead>
<tr>
<th>Product ID:</th>
<th>Product name:</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Developed LOC</th>
<th>Actual</th>
<th>Estimate deviation (+/- %)</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Generated LOC</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Reused LOC</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Number of generated screens</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Function Point</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Number of produced report</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Number of database table</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Number of incoming transaction type</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Number of outgoing transaction type</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Number of internal transaction UI</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Number of external transaction UI</th>
</tr>
</thead>
</table>

#### Error Profile:

<table>
<thead>
<tr>
<th>Phase</th>
<th>Number of error found</th>
<th>Number of error corrected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integration test</td>
<td></td>
<td></td>
</tr>
<tr>
<td>System test</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Acceptance test</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Maintenance</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Evolution Status:

<table>
<thead>
<tr>
<th>Total evolution occurrences</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Evolution occurrences per month</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Actual cost per evolution occurrence</th>
</tr>
</thead>
</table>

Figure 9.6: Final Project Status Form
situation. We introduce ten error categories which are expected to occur frequently in development projects. Figure 9.7 depicts the revised version of the Error Report.

Selected Projects for Case Study

In cooperation with experienced personnel at XXX, we have selected four completed projects (referred to as A, C, D, E) for data collection (activity A2) and one up-starting project (referred to as B) for monitoring (activity A3). The selected projects meet the required criteria stated earlier in 9.4.3. Four completed projects were carried out in the last two years. The person who helped us with the selection, is convinced that the involved project managers in selected, completed projects are clever to collect and document project information. The selected projects are of both development and up-grading types, and have both external and internal customers. The number of selected projects is determined to accommodate the available time frame we have for the case study.

9.5.2 A2 - Data Collection from Completed Projects

The activity on data collection from completed projects has also completed. We have spent considerably greater amount of effort than originally expected. This is partly due to the complexity of the document structure, and the large number of reports which must be examined. Another reason is that we didn’t succeed to invite any student to work on this task. The author has therefore carried out the activity by himself. In addition, when unclear thing are detected and thus need clarification, the involved persons either are not available or do not recall the particular incidents well. The new forms has been great help to structure collected data in a systematic manner. The obtained results during this activity are presented in section 9.6 under projects A, C, D and E.

9.5.3 A3 - Monitoring Up-starting Project

The activity has been of greatest interest and beneficial learning value. Such an activity offers us opportunity to gain valuable insight into and experiences on how to actually work in a real software project. The introductory section had gained positive feedback, and heartedly welcomed with promising enthusiasm. Despite of difficulty, we found it pleasant to experience that our conceptual framework is adaptable to a particular industrial case. The new paper-based forms have been installed into project directory in the file system. They are thus available to any participants of project B. However, they find the registration process cumbersome and time-consuming. They have to make a copy of a form, fill out and then store it in the same directory. We therefore face with the problem of naming different forms being collected. We make then a convention for naming forms to distinguish different filled-out forms. The name of the filled-out form is constituted by initials of author name and the recording date. Still, the project participants feel that they unnecessarily spend too much time for such registration activity, and the motivation is gradually declining. This observation has triggered us to implement a simple registration tool for process evolution which is described below in activity A4.2. The Error Report (depicted in figure 9.7) has unfortunately not been installed and used to to classify pre-release errors in project this up-starting project. Due to scarce resource and time, the collected data are not properly verified by the project manager. The obtained results of activity A3 are reported in section 9.6 under project B.
## ERROR REPORT

<table>
<thead>
<tr>
<th>Document name or module name</th>
<th>Number of error found / Number of error corrected</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Error Category:

- A: Requirement incorrect
- B: Design error
- C: Syntax error
- D: Logic, control, sequence error
- E: Assignment, computation error
- F: Interface, data access error
- G: Constants, initialization error
- H: Database error
- I: Communication error
- J: Other: ..............................................

### Test Method

- ☐ Code inspection done by author/developer
- ☐ Code inspection done by other
- ☐ Debugging
- ☐ Tested with own test driver
- ☐ Tested with other modules
- ☐ Other: ..............................................

### Detection effort

- ☐

### Correction effort

- ☐

**Further comments:**

---

**Figure 9.7: Error Report**
9.5.4 A4 - Tool Provision

This activity has been completed. Their associated results are fully described in chapter 8 under implementation issues, but also summarized here for the sake of comprehension.

The student group has implemented an experience database prototype, called EDBN (Experience Database for ...) for XXX. EDBN is implemented in MS-Access, and concentrates on structuring and storing existing metric data at XXX. It also provides support to search among completed projects based on similarities of a set of simple project and product profiles (e.g. name of project manager, customer, product type, ...). EDBN also generates Status Report, Progress Report and part of Experience Report which were previously produced manually.

As we carry out activity A3 to monitor the up-starting project B, it is emerging a need for a simple tool to record process evolution. The problems in using paper-based Evolution Registration Form is mentioned above. We have therefore implemented a Simple Evolution Registration Tool (referred to as ERT) in MS-Access. This tool visualizes Project Characterization Form (figure 9.4) and Evolution Registration Form (figure 9.5), and stores collected data in a simple database. ERT provides only simple browsing between collected process evolution within a project. ERT has eased the registration work, and encouraged participants to record process evolution.

A summer student has integrated EDBN and ERT into one single environment under more advanced ORACLE database. This integrated system (referred to as EDBN++) mainly extends EDBN with features to collect process evolution data. The identified project and product measures are also fully incorporated. EDBN++ provides support for multi-user access, and better graphical user interface. The choice to develop EDBN++ upon ORACLE database coincides with XXX's decision on platform usage in the future. Unfortunately, the EDBN++ has not been in active use. This is due to the limited resource both at our group and at XXX to follow up this activity after summer 1996. However, a new student project is now initiating to record collected data, enhance its user-interface to support World Wide Web, and actually use EDBN++ for planning new projects.

The EPOS extension work which is described in chapter 8, consists of two parts:

1. An experience database has been designed and implemented. Collected data on project, product measures, and process evolution from the case study are actually stored in the experience database. A set of associated tools are also implemented to record and analyze data. The results which are presented in section 9.7 below are derived by such tools.

2. A set of basic operations which dynamically manipulate task network layout, schedule and resources are also implemented. Those mechanisms will be validated against observed process evolution at XXX to see if they are capable to provide necessary corrective actions (see section 9.8 below).

9.5.5 A5 - Reuse of Experience Data

Unfortunately, we have neither time nor resource to complete this activity which seems to be of great importance to validate our work. On the other hand, we only manage to finish the planning phase at the time of writing this thesis. A pilot project has been selected and planned to start from the beginning of December of 1996 and finish in February of 1997. This would give sufficient time to include the
experiment results in this chapter. A concrete plan for the experiment has been elaborated, and distributed to the involved project manager. Due to some re-prioritizing at XXX, we got informed at the end of December that the project has been postponed. The fact that the author, who undoubtedly plays a key role in this experiment, needs time to write the thesis, selection a new project is almost out of question. The planned activities include:

- Record collected data on project properties and process evolution in EDBN++ at XXX. The retrieved analysis data from EDBN++ will be compared to those from EPOS. By doing so, we validate the correct behavior of EDBN++.

- The project manager will actively use EDBN++ to search for project similarities against completed projects.

- The initial project estimates are then revised to incorporate new insights and knowledge.

- The project is tracked and gained experiences are packaged to EDBN++.

Without such an activity to validate our obtained baselines, XXX is not able to make any decision on institutionalizing the technology we have applied in the case study. That is, the new elaborated forms have not yet been integrated into the XXX’s Quality System. However, XXX is very interested in the collected and analytical data so far and as mentioned above, our group and XXX are planning to follow up this work with a new student project in Spring 1997.

9.5.6 A6 - EPOS Validation

The description of this activity is dependent on the collected case study results which are presented in sections 9.6 and 9.7. We have therefore postponed the discussion of this activity to section 9.8.

9.6 Empirical Results

The collected data have come from two different sources. The first source is the data archaeology of the four completed projects while the second one is from monitoring the up-starting project. In common, they have gathered useful information on project/product characteristics, evolution and error during those studied projects. Such information have been stored in EPOS experience database to facilitate presentation, comparison and analysis. In this section, we present collected data from the five projects in the case study sample (referred to as A-E) in tabular and diagrammatic forms. Project B is the up-staring project for monitoring activity A3. The data presentation is divided into three groups: the project profile; evolution status and the project outcomes.

9.6.1 Project Profile

Following information are retrieved at project start by filling out the Project Characterization Form. Those measures are thus essentially fetched from the XXX’s Project Plan and Start Report of the studied projects. The project profile measures presented in the table 9.1 below are only a subset of those
Table 9.1: Project Profiles for five studied projects

included in the Project Characterization Form in figure 9.4. We only select those project profile measures which demonstrate obvious differences between the five studied projects.

We see from the table that most common project types and sizes at XXX are fairly represented in the project sample. In addition, the degree of risk level which is represented by competence availability, knowledge of applied technology and project difficulty, also cover the entire range of value domains. Projects C and E do not have a fixed number of participants, and personnel are constantly changing during the project life time. That is why their number of project members is defined as varied. Product profile is not included in the table because they are partly not of relevance in our further analysis, and partly lacking in the project archive.

9.6.2 Evolution Status

In this section, we present empirical results on evolution which are collected during the case study. A thorough analysis of such data is described in section 9.7. Most information on process evolution are retrieved from from the monthly Status Reports of the five studied projects. Table 9.2 illustrates the total number of evolution occurrences and their associated correction effort. The average frequency of evolution occurrences per month and average cost for each evolution are then derived.

Table 9.2: Evolution Profiles for five studied projects

Due to lack of evolution data in the project archive, some slots in the table above must be left empty. Complete information on process evolution during project E, and evolution cost in projects C and D are not documented at all. Despite of scarce information, the empirical results above demonstrate an average frequency of evolution occurrences of 2 or 3 per month. An average cost for correction effort is about 90 hours per process evolution. Only projects A and B have fully documented the impacts of process evolution. Therefore, we can hardly make further analysis based on such weak data foundation. However, we can demonstrate where process evolution comes from, i.e. where-dimension in the
9.6. Empirical Results

categorization framework. Table 9.3 presents the percentage of different evolution origins from five studied projects, while figure 9.8 depicts their average values. The figures obviously indicates a superior dominance in percentage of evolution which comes from customer. This conclusion corresponds to the intuitive perception at XXX. Only now, we have established a quantitative indication to where process evolution comes from.

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customer</td>
<td>34.9</td>
<td>73.6</td>
<td>47</td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>Sub-contractor</td>
<td>4.8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Executive</td>
<td>7.2</td>
<td>5.3</td>
<td>17.7</td>
<td>0</td>
<td>25</td>
</tr>
<tr>
<td>Project manager</td>
<td>33.7</td>
<td>15.8</td>
<td>5.9</td>
<td>20</td>
<td>25</td>
</tr>
<tr>
<td>Project member</td>
<td>19.3</td>
<td>5.3</td>
<td>29.5</td>
<td>30</td>
<td>0</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
</tbody>
</table>

Table 9.3: Percentage of Evolution Origin from five projects

![Pie chart showing percentages]

Figure 9.8: Average percentage of Evolution Origin

9.6.3 Project Outcome

In this section, we present the project estimates, and their actual status which are collected by the Project Characterization Form and Final Project Status Form. We then derive the estimation capability of XXX by comparing initial project estimates and actual performance. The baselines for effort distribution and error profile of the five projects are also presented.
9.6.3.1 Estimation Capability

During the case study we have collected both estimated and actual values of total project effort and duration. An estimation capability reflects the accuracy of estimates which are made at project start. An estimation capability index (ECI) for effort at XXX is thus defined by following formula:

\[
EstimationCapabilityIndex = \frac{ActualEffort - EstimatedEffort}{EstimatedEffort}
\]  

(9.1)

A positive ECI means under-estimation, while a negative one is interpreted as over-estimation of actual effort. That is, ECI prefers to be kept as low as possible. The ECI is converged to zero as the estimates become accurate. We apply equation 9.1 on collected data during the case study. By applying the collected values for actual and estimated effort, the ECI for five studied projects can be calculated. Table 9.4 illustrates the derived ECIs and the degree of delay from the studied projects.

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actual Effort (hour)</td>
<td>14364</td>
<td>6393</td>
<td>2915</td>
<td>2159</td>
<td>2760</td>
<td>-</td>
</tr>
<tr>
<td>Estimation Capability Index</td>
<td>1.05</td>
<td>0.68</td>
<td>1.54</td>
<td>0.46</td>
<td>0.82</td>
<td>0.91</td>
</tr>
<tr>
<td>Actual Duration (month)</td>
<td>30</td>
<td>16</td>
<td>6</td>
<td>4</td>
<td>7</td>
<td>-</td>
</tr>
<tr>
<td>Delay (day)</td>
<td>494</td>
<td>219</td>
<td>35</td>
<td>2</td>
<td>81</td>
<td>166</td>
</tr>
</tbody>
</table>

Table 9.4: Estimation Capability

From the table 9.4, we can see that the average ECI for effort is currently 0.91 at XXX. I.e. the actual project cost is almost two times higher than the initial cost estimate. It can also be interpreted that the average project effort overrun at XXX is currently 91%. It implies that the existing estimation method needs to be improved considerably to overcome such poor estimation capability.

The equation 9.1 is defined for easily deriving the actual effort value from a given estimates and a current ECI by applying following formula:

\[
ActualEffort = InitialEstimatedEffort \times (EstimationCapabilityIndex + 1)
\]  

(9.2)

The equation 9.2 can be used as a simple algebraic effort prediction model. That is, an initial estimate is made by using existing estimation method. The accuracy of such an estimate is improved by applying the equation 9.2 with the current ECI of the company. The quality of derived “estimated” actual effort is strongly dependent on the accuracy and validity of the ECI. The ECI for effort certainly becomes more accurate as new completed projects provide their estimated and actual values for further revision and refinement. In the following, we will explain how the accuracy of ECI can be improved by increasing its level of granularity.

As table 9.4 shows the ECIs for the entire projects, figure 9.9 and table 9.5 on the other hand illustrate ECIs associated to a particular activity in development life cycle of five studied projects. The average ECIs for both effort underestimation and overestimation for such activities are also represented in the table.

Neither under- nor over-estimation is desirable in estimation software projects. However, effort underestimation for a given activity causes serious problem and is thus costly to correct. The table 9.5 illustrates that planning, integration test and documentation are three activities which have greatest underestimation ECIs. High ECI in planning can be a direct consequence of either difficulties in contract
Figure 9.9: Under- and over-estimation ECI associated to life-cycle activities
negotiation with customer, or the poor methodological support in elaborating project plan. Frequent revision of customer requirements during project execution implies many re-work of the planning activity. The high under-estimation ECI value for integration test activity is probably caused by lack of error baselines as well as testing efficiency baseline from prior projects. It implies that a larger number of errors are detected and take longer time to remove than initially predicted. The problem with high under-estimation ECI in documentation activity is probably due to the low priority that is often given to such activity when there is limited project budget.

The reason for introducing both under- and over-estimation ECIs associated to a particular life cycle activity is to improve their accuracies when using in effort prediction equation 9.2. In stead of using the company’s average ECI (i.e. 0.91) as illustrated in table 9.4, the project manager can use the ECI values associated to a particular activity in table 9.5. The project manager must initially select a set of completed projects which are most similar to his/her up-starting project. If such selected, completed projects tend to either under- or over-estimate effort for a given activity, the corresponding ECI value must be used. We illustrate with a simple example. A new project, called Z, wants to estimate the effort which is required for integration test. Assume that the external project profiles of Z are similar to those of completed projects A and B. By studying table 9.5, we see a stronger tendency of under-estimation in A than of over-estimation in B. The under-estimation ECI for integration test (i.e. 1.42) is thus used. The project manager initially plans to use e.g. 150 hours in integration test for Z. By applying the effort prediction equation with the selected ECI, we have:

\[ Newestimate = 150 \times (1.42 + 1) = 363 \text{ hours} \]

It means that the initial estimate for integration test should be revised to 363 hours in the project plan. However, the accuracy of the new estimate is considerably improved if there exist many data points from similar projects. If an appropriate ECI is not easy to select from completed projects, the set of similar projects must be reduced. This can be done by specifying more constraints to the project profiles, or associating weights to project profile measures (see chapter 7).
9.6. Empirical Results

9.6.3.2 Actual Effort Distribution

In this section, we present a baseline for effort distribution over life cycle activities based on collected data during the case study. Table 9.6 illustrates actual effort distribution in percentage over phases of development life cycle for five studied projects and their average values. Figure 9.10 diagrammatically shows the average effort needed for each development phases.

<table>
<thead>
<tr>
<th>Activity</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Planning</td>
<td>2.7</td>
<td>-</td>
<td>17</td>
<td>4.3</td>
<td>-</td>
<td>4.8</td>
</tr>
<tr>
<td>Project administration</td>
<td>14</td>
<td>15.5</td>
<td>13</td>
<td>15.3</td>
<td>6.9</td>
<td>12.9</td>
</tr>
<tr>
<td>Analysis</td>
<td>2.3</td>
<td>6.4</td>
<td>-</td>
<td>7</td>
<td>7.3</td>
<td>4.6</td>
</tr>
<tr>
<td>Design</td>
<td>6.3</td>
<td>10.8</td>
<td>6.4</td>
<td>14.6</td>
<td>14.5</td>
<td>10.5</td>
</tr>
<tr>
<td>Realization</td>
<td>41</td>
<td>34.5</td>
<td>33</td>
<td>36.8</td>
<td>43.5</td>
<td>37.8</td>
</tr>
<tr>
<td>Integration test</td>
<td>7.6</td>
<td>2.5</td>
<td>17</td>
<td>4.4</td>
<td>3.8</td>
<td>7.1</td>
</tr>
<tr>
<td>System test</td>
<td>8.7</td>
<td>13.8</td>
<td>4.1</td>
<td>8.8</td>
<td>7.6</td>
<td>8.6</td>
</tr>
<tr>
<td>Acceptance test</td>
<td>6.4</td>
<td>1.5</td>
<td>1.6</td>
<td>8.8</td>
<td>7.5</td>
<td>5.2</td>
</tr>
<tr>
<td>Pilot</td>
<td>0.8</td>
<td>5</td>
<td>1</td>
<td>-</td>
<td>8.3</td>
<td>3.4</td>
</tr>
<tr>
<td>Documentation</td>
<td>2.3</td>
<td>1.6</td>
<td>5.2</td>
<td>-</td>
<td>0.6</td>
<td>2.0</td>
</tr>
<tr>
<td>Other</td>
<td>7.9</td>
<td>7.7</td>
<td>1.7</td>
<td>-</td>
<td>-</td>
<td>3.1</td>
</tr>
<tr>
<td>Total</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
</tbody>
</table>

Table 9.6: Actual Effort Distribution over life cycle activities for five projects

The values in table 9.6 present the actual effort which is necessary to complete a particular life cycle activity. From the pie chart, we can also see that small effort percentage spent in requirement analysis results in major work in realization. Such observation indicates a need to put more emphasis to analysis to reduce the implementation effort. The average percentage of effort in figure 9.10 can be used to schedule necessary effort for activities based on an initial project budget. Of course, the actual effort distribution for a specific project can also be used in such scheduling. As mentioned above, the more projects are executed and their data are collected, the more precise the ECI and initial estimates on effort become.

9.6.3.3 Error Profile

Below we present a baseline for pre-release error which are derived from collected error data from project A and B. Other three completed projects did not collect and document such error data. Figure 9.11 depicts the number of pre-release errors and their total removal effort during the test phase. The required effort to remove each error is also calculated and illustrated in the figure. The total error removal effort during acceptance test is only available for project B, and is therefore discarded in the figure.

The post-release error data of each project is not considered in this case study due to the fact that such data can only be retrieved from other places than System Division to which the case study is conducted. Besides, this is not defined as objectives in the case study. The figure 9.11 shows that most errors are found in system test. The increase in error removal effort from integration to system test emphasizes
the fact that it takes longer time to remove errors as they are detected at later phases in development life cycle. For comparison, the removal effort of a post-release error is of 7 hours (see section 9.3). It is also well recognized that error profile is strongly influenced by product profile measure (e.g. product type, complexity, size, modularity, ...). Since XXX does not record such product metrics, it is difficult to provide normalized error density (i.e. number of error per 1K lines of code), or various interesting related metrics on a given product. However, this work on collection of error data which are summarized in figure 9.11 clearly demonstrates the benefit on having such error baselines to predict necessary testing effort. This work has also helped XXX to realize the importance of such error data, and then put more emphasis on error collection and analysis (c.f. the discussion on the new error report above in section 9.5).

9.7 Evolution Analysis

In this section, we describe the results which are obtained by analyzing collected process evolution data. The analysis is basically performed within the context of our categorization framework. A set of typical process evolution patterns are then identified. Empirical relations between process evolution and project profile are also recognized.

We introduce below two ways to perform analysis by combining arbitrary evolution dimensions in the categorization framework (i.e. where, why, what, how, when and by-whom). The following analysis are done by combining dimensions where-why and where-how. The reason for our choice of such combinations is simply that the analytical results demonstrate interesting findings, and are further discussed in subsequent subsections. Of course, there is no restriction on which or on number of evolution
dimensions to be combined to perform the analysis.

The numbers which are reported in following tables, are derived from our Evolution Analyzer Tool (see chapter 8). Only four greatest process evolution origins in figure 9.8 (i.e. Customer, Executive, Project Manager and Project Member) are taken into consideration in the analysis.

### 9.7.1 Where-Why Frequency Analysis

The percentages in table 9.7 are derived by keeping one Where-category fixed (e.g. Customer), and then varying the category values in Why-dimension. That is, they illustrate frequency of evolution occurrences (in percent) distributed over different cause categories. Only interesting categories in the Why-dimension are selected for illustration. The most frequent evolution patterns are identified and discussed in depth in section 9.7.3.

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Customer</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ambiguity</td>
<td>6.5</td>
<td>5.3</td>
<td>17.7</td>
<td>11.1</td>
<td>0</td>
<td>8.1</td>
</tr>
<tr>
<td>Error</td>
<td>1.3</td>
<td>5.3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1.3</td>
</tr>
<tr>
<td>Requirement revision</td>
<td>5.2</td>
<td>15.8</td>
<td>17.7</td>
<td>44.4</td>
<td>25</td>
<td>21.6</td>
</tr>
<tr>
<td>Lack of competence</td>
<td>3.9</td>
<td>5.3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1.8</td>
</tr>
<tr>
<td>Delay</td>
<td>15.6</td>
<td>10.5</td>
<td>5.9</td>
<td>11.1</td>
<td>25</td>
<td>14</td>
</tr>
<tr>
<td>Postponement</td>
<td>5.2</td>
<td>21</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5.2</td>
</tr>
<tr>
<td>Lack of resource</td>
<td>1.3</td>
<td>0</td>
<td>5.8</td>
<td>0</td>
<td>0</td>
<td>1.4</td>
</tr>
<tr>
<td><strong>Executive</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Re-prioritizing</td>
<td>7.8</td>
<td>5.3</td>
<td>17.7</td>
<td>0</td>
<td>25</td>
<td>11.2</td>
</tr>
<tr>
<td><strong>Project manager</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Under-estimate</td>
<td>29.9</td>
<td>15.8</td>
<td>5.9</td>
<td>11.1</td>
<td>25</td>
<td>17.5</td>
</tr>
<tr>
<td><strong>Project member</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Error</td>
<td>11.7</td>
<td>0</td>
<td>17.6</td>
<td>11.1</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>Lack of competence</td>
<td>5.2</td>
<td>5.3</td>
<td>5.9</td>
<td>0</td>
<td>0</td>
<td>3.2</td>
</tr>
<tr>
<td>Delay</td>
<td>3.9</td>
<td>0</td>
<td>0</td>
<td>11.1</td>
<td>0</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 9.7: Frequency of Evolution Occurrences distributed over Causes

### 9.7.2 Where-How Frequency Analysis

On the similar manner, the percentages in table 9.8 are derived by keeping one Where-category fixed (e.g. Customer), and then varying category values in the How-dimension. That is, they represent the frequency of evolution occurrences (in percent) distributed over different impact categories. Only some interesting categories in the How-dimension are included in the table. The identified impacts are used to validate EPOS's approaches to manage process evolution in section 9.8.3.

The table 9.8 shows that rework and prolong activity are superior evolution impacts on project schedule. Such impacts require additional allocation of resources according to initial plan. The frequency of evolution which requires rework, constitute 23%, while 36% of total evolution occurrences imply
additional resources allocated to a prolonging activity. A postponed activity due to a process evolution does not involve any additional cost. However, such evolution imply that the plan must be re-scheduled and allocated resources must be released. Still depending on the extent of resource needed to such rescue activities, the figures in the table clearly demonstrates the importance and seriousness of process evolution.

9.7.3 Typical Evolution Patterns

Based on the figures presented in tables 9.7 and 9.8 above, we identify eight most typical and frequent evolution patterns. Such evolution patterns are below listed according to decreasing frequency percentages. Within the description of evolution pattern, we also emphasize the necessary corrective actions. Such rescue actions will be then realized by invoking different operations to manipulate project plan (i.e. task network) in EPOS (see section 9.8.3).

1. **Customer revision (21.6%)**: The customer issues new or revised requirements according to the initial specification. Such changes requires that completed activities must be re-executed. The extent of rework depends on which phase in development life-cycle the requirement change is requested. New activities can also be added into the project plan to deal with new or enhanced system functionalities. Note that the requirement revisions after initial delivery (i.e. maintenance phase) have not been taken into consideration in this case study.

2. **Under-estimation (17.5%)**: The project manager tends to under-estimate various activities under planning. This is partly due to misunderstanding of user requirements, and partly lack of empirical foundation for making estimates. Such under-estimation usually causes that initial estimates must be revised, and project plan must be re-scheduled. Sometimes, new activities are introduced while existing ones are either removed, divided or merged.

3. **Customer delay (14%)**: The customer delays to deliver documents (e.g., requirement, customer
site interface) as initially agreed. Such delays require a postponement of activities which are dependent on the given deliverables. The project plan must then be revised to reflect that fact.

4. **Resource re-allocation (11.2%)**: The executive management (division or senior manager) re-allocates project personnel to other project as a result of strategic re-prioritizing. Such happen when they want to “rescue” other urgent projects, or to satisfy a higher prioritized customer request. This high frequency of staff turn-over results in a violation of the project plan. That is, necessary resources and competence are not available to keep in project progress on schedule. Human factors associated to a particular activity must be removed, and other dependent activities must therefore be postponed.

5. **Ambiguous requirement (8.1%)**: The customer is not clear when the initial requirements are specified. This is due partly to their limited insight into the problem, and partly due to their lack of competence in a given technology. That is, important requirement details are neglected or vaguely defined. Such ambiguous requirements usually imply a rework of requirement specification, or an insertion of corrective activities.

6. **Error commitment (8.0%)**: The project member have injected errors in technical documents. Such an evolution pattern often initiates either rework or prolong the current activity. An specialized error removal task can also be inserted to deal with the problem. Such corrective actions imply delay and thus re-scheduling of the project plan.

7. **Customer postponement (5.2%)**: Unexpected requests for postponing the start date of an activity are issued by the customer. Such postponement causes that the initial project plan must be revised. The time and human factors which are associated to scheduled activities are changed.

8. **Lack of competence (5.0%)**: This phenomena is more common among project member (3.2%) than among customer (1.4%). Such events are resolved by assigning project personnel to training to improve e.g. their knowledge on a particular development method/language. Training activities are thus added to the existing project plan.

Based on the identified evolution patterns above, we have suggested a set of improvement initiatives which are described in section 9.9.

### 9.7.4 Empirical Relations between Evolution and Project Profile

In this section, we present our recognition of empirical relations between collected data (in section 9.6) and project characteristics (see table 9.1). In particular, we observe relations between frequency of evolution occurrences, estimation capability indices (ECIs), and typical evolution patterns on one hand, and external project profiles on the other hand. Such empirical relations improve our ability to predict and then anticipate process evolution based on what we know at project start (i.e. project profile measures). We have revealed following interesting relations. Of course, this list is not exhaustive.

**Project type versus Evolution frequency** Development projects (A and C) experience more turbulence than upgrading ones (projects B, D, and E). This fact is illustrated by the high percentages of evolution frequency in table 9.2 in projects A and C. This phenomena can be explained by the fact that both customer and XXX in development projects deal with a new problem which they often do not have sufficient technical competence.
9.7. Evolution Analysis

Project type versus ECI  The table 9.4 illustrates higher ECIs in development projects than upgrading ones. In addition, the development projects have stronger tendency to under-estimate activities in development life cycle than upgrading ones. This fact is obviously illustrated by ECIs of critical phases such as Realization and Integration test (see 9.5). Again, the limited insight into the solution space often lead to poor effort estimates.

Project type versus Customer revision  A high degree of customer revisions is found in upgrading projects (B, D, E) in table 9.7. This observation is somehow hard to explain. There are two assumptions usually associated to upgrading projects. Firstly, technology is well-known, and required competence is available. Secondly, the customer often states clear and well-defined enhancement requirements as they have been using the product for a period of time. However, the second assumption does not hold in our case. For instance in project B, the customer at project start delivered an requirement specification which has thereafter been revised so many times that the initially agreed-upon contract must be completely re-negotiated. Therefore, this empirical relation must be evaluated carefully later as more data points are collected.

Project duration versus Under-estimation  The degree of under-estimation seems to increase proportionally with the project duration (see table 9.7). It is easy to accept the fact that it is hard to predict the behavior of some activities which are going to take place far beyond in the future. Moreover, project managers at XXX suffer from the lack of empirical baselines from prior projects to rely their estimates on.

Project duration versus Customer delay  The extent of customer delay is proportional to the project duration (see table 9.7). This is probably due to the fact that customer tends to give such projects lower priority, and thus does not respect the initially agreed-upon deadlines.

Project type versus Ambiguous requirement  Customers in development projects are often uncertain on what features they really need. This normally leads to unclear and vague specification of initial user requirements.

Project type versus Error commitment  Table 9.7 shows a superior representation of errors done by project members in development projects. This is perfectly natural when the project members must address a problem to which they do not completely master.

Customer type versus Customer revision  Development project E with an internal customer (i.e. XXX's Product Division in this case study) has experienced a higher degree of requirement revisions than other projects. This relation is not quantitatively illustrated by the collected data material, but has been acquired through reading other reports. Nevertheless, it is easy to understand when we accept the fact that internal development projects do not bear any economical risk for XXX, and in addition are often given lower priority by the executive management level.

It is worthwhile to stress that the empirical relations above are retrieved by our limited data ground which are collected during the case study. The explanations for the relations are thus somehow speculative. As the amount of collected data grows, more valid relations can then be extracted. After all, our primary intention is to demonstrate the usefulness and benefits from collecting and analyzing such information.
9.8 EPOS Validation

In section 9.4.3, we have planned an activity whose intention is to use case study results to validate EPOS's approaches and mechanisms in managing process evolution. The main idea is to obtain realistic evolution data from software company and export them to our PSEE prototype EPOS. By doing so, we can both validate the applicability of our current work, and motivate further improvement. Figure 9.12 illustrates activities involving in such a cooperation from which both EPOS and XXX can mutually benefit.

![Diagram showing process interactions between EPOS and XXX]

**Figure 9.12: Case study at XXX and EPOS**

From figure 9.12, we can identify four major tasks (enumerated from 1-4) which are planned to be performed:

1. **Modeling**: Relevant parts of XXX's process are modeled in EPOS-PM to facilitate for experiments and validation of our research ideas in the future.

2. **Packaging**: The collected data in the case study are exported to and stored in EPOS. Such empirical baselines will be exploited by our tools for further analysis and assistance project execution.

3. **Validating**: EPOS basic mechanisms in manipulating task network, schedule and resources are validated with the identified evolution patterns from the case study. That is, we demonstrate how the corrective actions associated to such evolution patterns are realized in EPOS.

4. **Suggesting**: Improvement initiatives are suggested to XXX for better manage process evolution based on our collected data, analytical results and our comprehensive insight into technology and
current best practices. In addition, we outline some enhancement suggestions to EPOS based on the gained experience of validation activity above. The description of this activity will be presented in section 9.9.

9.8.1 Modeling XXX’s Quality System

To be able to simulate a XXX’s project context, the project model and system development model (depicted in figures 9.1 and 9.2 respectively) are described in EPOS by our PML, SPELL. Our main objectives of this modeling activity are:

- Regarding as another process exercise to be modeled in EPOS, similar to our previous modeling experiences (e.g. ISPW67, Review-example) which are reported in chapter 5.

- Facilitating our further internal research experiments on the topic of process evolution. The experience database in EPOS is enriched by continuously store project data from XXX. Frequent evolution patterns are recognized. Several crucial relations are found out to continuously improve the predictability at XXX (organizational learning).

- Justifying our research by clearly demonstrating the relevancy and applicability of our work against external, industrial cases. In addition, such an effect surely strengthens our research position, and thus attracts more students.

Necessary types have been defined in EPOS to describe the selected process models from XXX. The process models (i.e. types) in EPOS are defined based on the description of the project model and system development model depicted in figures 9.1 and 9.2. Figure 9.13 depicts an simplified example of a task network representing a simulated project_x at XXX.

The project_x has a mandate as input and produces a final family system together with an experience report. The project is then decomposed into activities: make_plan, realize and close_up as described in XXX’s project model in figure 9.1. The XXX’s system development model description is used to model the task network under realize. They include analyze, design, implement and test. A design document is produced and then reviewed for each module in the family system. Each module is then tested separately before the entire family undergoes the testing phase, including integration, system and acceptance test routines.

This modeling activity is obviously unnecessary if we and our students can freely get access to collected data at XXX. Unfortunately, this is not the case due to strict security regulations at XXX. An experiment or case study can not be initiated unless it clearly demonstrates benefits for XXX. Moreover, there is seldom available resources at XXX to assist and follow up the such experiments. That is something we have painfully experienced in this case study.

9.8.2 Packaging Empirical Data

The data collected from the case study are exported to and packaged in EPOS. Process evolution occurrences are categorized and recorded in our EPOS experience database. Our new process tools (see chapter 8) are used to analyze the collected data. In fact, the analytical results which are reported in
sections 9.6 and 9.7, are derived from EPOS experience database and by using our process tools. Currently, EPOS experience database only contains information from five studied projects. However, adequate tools are already provided for recording data from new projects. By performing this packaging activity, we are convinced that EPOS is capable to classify, record, and analyze process evolution data from a real software company.

9.8.3 Managing Typical Evolution Patterns in EPOS

When presenting typical evolution patterns in section 9.7.3, we also describe the necessary actions which must initiate to correct a particular evolution pattern. Most of them imply a revision of the project plan (i.e. task network in EPOS) by either adding new activities, manipulating existing ones, changing schedule properties of activities.

A detailed description of how such operations are implemented in EPOS to manage task network evolution are found in chapter 8 under section 8.4. Below we summarize basic operations that can be used to manipulate the EPOS task network:

Task removal Remove an existing task from the task network (in section 7.5.1.2).

Task insertion Put an additional task to the existing task network (in section 7.5.1.1).

Schedule revision Start time, Elapse time and Duration of the task in the task network can be revised during execution (in section 7.5.2).
9.9. Improvement Suggestions

Resource revision Performer of the task in the network can be replaced. As the consequence, the task is either suspended or canceled (in section 7.5.2).

Table 9.9 shows how eight evolution patterns (enumerated 1–8 according to the list in section 9.7.3) are dealt by EPOS’s approaches to manage task network evolution (bottom-up process model evolution).

<table>
<thead>
<tr>
<th>Evolution pattern</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task removal operation</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Task insertion operation</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Task schedule revision</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Task resource revision</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

Table 9.9: Evolution Patterns versus Approaches in EPOS

9.9 Improvement Suggestions

In this section we state our suggestions for improvements towards the studied software company XXX and EPOS on the basis of case study results and findings.

9.9.1 Suggested Improvement Initiatives for XXX

By consolidating the findings and analytical results from the case study, we propose following suggested improvement initiatives for XXX in the future. The following list is not sorted in any priority order.

- The most frequent evolution pattern is revealed to be user requirement revision. XXX should have therefore better dialog with customer in requirement specification phase. That is, advanced technology in requirement engineering can be adopted to better understand/structure requirements and thus prevent them from modifying later.

- The second most frequent evolution pattern is concerned the issue of under-estimation. This problem can be remedied by adopting better estimation method. The case study has demonstrated the benefit of learning from past experiences to establish baselines and then improve estimate accuracy. The existing quantitative baselines at XXX are still insufficient and incapable to provide precise estimates. However, they will be improved over time if XXX commits to pursue the same work direction. On the other hand, XXX should initiate better cooperation between System Development Division and Sale Division in contract bidding process. That is, realistic but also competitive bids must be elaborated together.

- We have also seen in 9.7.4 that there exist empirical relations between long project duration on one hand and under-estimation and customer delay on the other hand. This problem can be avoided if XXX adopts either incremental or prototyping approach in developing software. That is, several product deliveries should be made during big projects to offer customer opportunities
for testing and eventually revising their requirements. The period of time between deliveries is reduced, and thus customer feedback is easily and cost-effectively incorporated.

- Incorporate the new paper-based forms for characterizing project/product (figure 9.4); collecting evolution data (figure 9.5); collecting actual project/product status (figure 9.6), and categorizing errors (figure 9.7) into appropriate handbooks in the Quality System. Specifically, the three first forms are included in the Project Management Handbook, while the last one in the System Development Handbook. By doing so, XXX institutionalizes an effective data collection process for the company. Since the EDBN++ needs times to be validated, those paper-based forms can be used in the meantime.

- Select one or two pilot projects which will use the EDBN++ for recording project/product measures and process evolution. In parallel, the collected data from the five studied projects should be stored in EDBN++ available for future projection. The reason to run EDBN++ together with pilot projects is to validate its applicability, user-friendliness and correctness. Necessary enhancements should be recognized and integrated before EDBN++ is released for use within the company.

- Project managers are encouraged to use the EDBN++ both to make project estimates, and to anticipate possible impacts during project execution (planning with contingency). It is a major step towards continuous learning and reusing past experiences. As many projects record their data into the EDBN++, the existing baselines gradually become valid, representative and convergent. The improvement evidences on estimate accuracy are then significantly demonstrated. Patience and full commitment are key issues to establish a valid experience foundation. We refer to an example of at NASA-SEL. It took them 18 years (1977-94) with several experiments to gather data before they manage to consolidate to a set of representative and reliable baselines.

- To be able to achieve significant improvement in error reduction during development, XXX is suggested to seriously collect and classify errors according to their types, origins, severities, and related characteristics (partly covered by the revised Error Report in figure 9.7). XXX can then obtain a deep insight into the error problem and prioritize corrective actions according to error impacts. Adequate technology can thus be evaluated and introduced to reduce number of errors or to detect them earlier. That is, required effort to remove them can be declined considerably.

### 9.9.2 Improvement Suggestions to EPOS

Based on our experience in validating EPOS in section 9.8, we outline below some improvement suggestions for further work (see also the discussion on further work in concluding chapter 10).

- The EPOS experience database should consider to use the same database as the EDBN++, i.e. ORACLE. It will thus imply easy access and exchange of experience data between XXX and EPOS. That also certainly save a great amount of effort to export data from EDBN++ and manually register them in EPOS experience database for experiment.

- The planning process in EPOS currently concerns only the product space. It can be extended to take into consideration also available resources, personnel and various constraints.
9.10 Evaluation and Discussion

The obtained results and gained benefits throughout the study are evaluated against the objectives in section 9.4.1. We also provide answers to the research questions which are formulated in section 9.4.2. A discussion of validity of the obtained results as well as lesson learned are then presented.

9.10.1 Gained Benefits versus Objectives

The case study benefits are evaluated against our initial stated objectives in 9.4.1.

O1. Getting baselines We have elaborated four paper-based forms to collect external project/product characteristics, process evolution, error categories, and actual/final project status. Those forms have been used to gather data from four completed projects. Three of those forms have been incorporated into the process to collect relevant data during an up-starting project. The data collection activity has been performed for almost one year. However, due to limited resource and time, we have not managed to collect data from several than five projects. The obtained baselines and their derived analytical data must therefore be viewed with respect to such poor data ground. Some important values from the obtained baselines are summarized below. In section 9.6, we have presented gained baselines for process evolution, estimation capability, effort distribution, and error profile during development project. The process evolution baseline illustrates that XXX's projects should expect to experience 2-3 process evolution per month, and each implies an additional cost of 90 hours. The baseline for estimation capability reveals that projects have an average effort overrun of 91%. On the other hand, XXX has recorded that only 26% of projects deliver on time. That is, 74% of projects have an average overrun of 91%. This is really scaring if they are correct figures. Section 9.7 has performed detailed analysis on process evolution to provide better understand on their origins, causes, and impacts. Based on such analysis, eight most frequent process evolution patterns are identified. Customer requirement revision, under-estimation, and customer delay together constitute about 53% of total evolution occurrences. Viewing the problem from another perspective, the evolution baselines reveal that 23% and 36% of process evolution require rework and allocation more effort to a prolonging activity respectively. Those figures above clearly illustrate the seriousness of process evolution and its impact to software projects. We have also observed eight empirical relations between external project characteristics on one hand and estimation capability, process evolution patterns on the other hand.

O2. Assessing baselines The obtained baselines plan to be used in estimating, planning and tracking a new pilot project. We have elaborated a plan for activities to be carried out. In addition, an effort prediction equation 9.2 has been made to improve the estimate accuracy. Unfortunately, the assessment of obtained baselines is postponed due to cancelation of the pilot project at XXX. Moreover, it is scarce of time to select a new pilot project to fulfill the assessment.
O3. Validating EPOS  The collected data have been exported and stored in EPOS experience database. In fact, the evolution analysis which presented in section 9.7 has been carried out in EPOS. Relevant parts of XXX’s process models have been described in EPOS facilitating for further experiments. The impacts of eight evolution patterns are handled by our approaches for manipulating layout, schedule and resources of the task network (i.e. project plan). It appears that our implemented task network manipulating operations manage to deal with the identified evolution patterns in a satisfactory manner.

9.10.2 Assessment on Stated Research Questions

The research questions which are stated in section 9.4.2 will be assessed based on the gained results in this case study.

Q1: We have seen that all collected process evolution occurrences imply impact on the project plan (i.e. enactable/enacting process model) rather than on the handbook of XXX’s Quality System (i.e. generic process model). It illustrates that it is more pertinent and realistic to dedicate more research effort to address such evolution. However, this observation may indicate either that XXX’s Quality System do not have any undesirable and inefficient properties, or that QS improvement is not considered as project participants’ major concern and responsibility. Anyway, we are convinced that there is higher rate of changes to the enacting process models than to the generic ones.

Q2: The case study results have shown that our categorization framework has been an useful means to categorize process evolution occurrences. The Evolution Registration Form which is specialized for the case study, manages to classify all process evolution incidents either retrieved from project archive or observed in the up-starting project. However, a few evolution occurrences are hardly classified by one single category within the same aspect, but is better described by a combination of two or more categories. E.g., a particular evolution may be caused by both project manager and project member. Of course, the number of category can be dynamically extended to remedy this problem. But we did not due to scarcity of time.

Q3: We have managed to identify some interesting empirical relations between external project profile on one hand, and estimation capability and evolution patterns on the other hand (refer to section 9.7.4). Those identified relations improve ability to anticipate process evolution and predict behavior of future projects based on external project characteristics (i.e. project profile measures).

Q4: Our proposed approach comprises of three phases which can be grouped into two main categories: experience learning (baselines establishment) and experience reuse (initial and iterative planning support). The planned activities for the case study which is described in section 9.4.3, have reflected those two categories.

1. The experience learning is represented by activities A1–A3 retrieve and establish baselines which are analyzed in section 9.7.

2. The experience reuse is planned to execute in activity A5 including both initial planning support (A.5.1) and iterative planning support (A.5.2).
9.10. Evaluation and Discussion

Q5: Approaches in EPOS to manipulate task network layout, schedule and resources are adequate to handle the impacts of identified evolution patterns. However, they still need to be validated with several cases before we can be fully convinced.

9.10.3 Threats to Validity

In this section, we discuss the validity of gained results in the case study. Following issues are regarded to be of importance when the case study results are assessed:

**Uncertain and missing data** Information in the project archive at XXX is not always complete. Many information and relevant measures which are required by the case study, can therefore not be retrieved. Involved persons do not recall or have only vague memory of the particular incident. Although the selected projects satisfy the criteria on well-documentation, we still miss some important values, e.g. both estimated and actual corrective cost associated to a particular process evolution. Both uncertain and missing data certainly diverge our obtained baselines.

**Small project sample** The number of selected projects in the case study sample is not many. Limited number of projects surely weaken the confidence and representativeness of the case study result. Therefore, the obtained results are not necessarily valid for XXX from a statistical viewpoint. It is due to our time limit and the scarce resource at XXX to assist the case study.

**Varied documentation skill** The ability to document and collect project data largely varies among project managers. This difference is obviously reflected by the degree of details in reported documents. The quality of the collected data is also varied from one project to another. This is probably due to the general recognition of little use of recorded experience data at XXX, and such data collection activity is therefore given low priority.

**Common perception** The personnel in the selected, up-starting project have often experienced a great deal of turbulence in their daily work. They therefore find it unnatural to report about a particular process evolution. That is, such mentality must be changed before we can absolutely capture all process evolution occurrences.

9.10.4 Lesson Learned

Below we give an outline of the lessons we have gained by conducting the case study. Some of them are of universal value for anybody who is going to perform a case study in the future.

- The usage of paper-based forms to collect data are not effective and time-consuming. Only a very simple collection tool (c.f. Evolution Registration Tool in 9.5.4) is sufficient to motivate people and to reduce the required effort. The increase in number of collected evolution forms after the installation of the registration tool are easy to notice. However, this observation is not peculiar if we imagine that their daily working situation is already full of paper sheets to be filled out. Bearing that thought in mind, the number of new forms in the case study should be reduced to the minimum.

- Although the idea of and motivation in conducting improvement activities are widely accepted at XXX, the executive management hardly gives such activities high enough priority. This lack
of commitment and resistance from the executive management level is probably due to the high work load and high number of project requests in company pipeline. The fear of losing competitive advantage in the market has been a major reason for giving up long term improvement activity.

- The performance of the case study, despite a numerous difficulties, has been characterized as an useful and informative experience for all of participants. For academics, it is an unique opportunity to get valuable experience, insight and most of all realistic, raw data for validating their work. On the other hand, software company can benefit from such studies in the sense that they get transferred new technology and current best practices which they have neither capacity nor competence to carry out otherwise.

9.11 Chapter Summary

In this chapter, we have presented a case study conducting against a software company which is one of the leading software providers for banking application in Norway. The case study has been conducted according to the Experimental Paradigm of Basili. We first provide a case study context in which background and current practices of the studied software company are described and assessed. The case study has been carefully planned by first determining its objectives and then discussing feasible strategies. We have further stated a set of pertinent research questions to which the case study result will provide answer. Six activities are planned to both establish existing baselines and validating them. Four new paper-based forms have been elaborated to collect project and evolution data. Experience databases and their associated tools have been implemented for XXX and EPOS. We have also in this chapter presented and analyzed collected data to provide XXX a set of baselines on estimate capability, effort distribution, error profile during development, and process evolution. In addition, we manage during analysis to recognize eight most frequent evolution patterns, and empirical relations between external project characteristic and process evolution. The collected data are also exported and stored in EPOS for validation. Selected parts of XXX’s process models are described in EPOS to facilitate internal experiments in the future. Approaches and implemented mechanisms in EPOS are validated to demonstrate its ability to react to the identified evolution patterns. Based on the gained results, we have suggested some improvements to both XXX and EPOS. Finally, we evaluate our results against both issued objectives and stated research questions. Issues which are of importance for the validity of the case study results and gained lessons are presented. It is worthwhile to note that the gained results in the case study are going to appear in the forthcoming conference proceeding of ICSE-97 [NWC97].

It is important to emphasize that this case study has by all means no intention to establish a valid and representative baselines for XXX in such a short period of time. On the contrary, this case study aims to demonstrate the importance and usefulness of past project experience to achieve project success in the future. It is widely recognized that it is impossible to generalize across projects due to their inherent differences. However, we believe that there exist patterns for both successful and failed projects. Our work therefore lays a necessary foundation to collect, analyze and reuse experiences to improve predictability in future projects. Further, the case study should be regarded as a starting point for a series of similar improvement initiatives focusing in managing process evolution. The obtained baselines are currently still immature and diverge. But, their validity will be improved over years as future projects are enriching them with several experience data points.
Chapter 10

Evaluation, Conclusion and Further Work

10.1 Introduction

In this chapter, we firstly summarize the entire thesis in section 10.2 followed by an outline of main achievements in section 10.3. In section 10.4, the highlights of the EPOS prototype are identified and then specifically evaluated according to the requirements stated in chapter 3. Thereafter, we compare our achievements with the existing PSEEs and current work in software evolution in section 10.5 before we outline some disadvantages of EPOS. Some further works are finally presented in section 10.7.

10.2 Summary of the Thesis

The thesis aims to address the problem of software evolution in software engineering projects within the context of a PSEE. We start with an investigation of state-of-the-art in chapter 2 on six central topics: software engineering (including software process and PSEEs); software process improvement models (ISO-9000 series, CMM, QIP, BOOTSTRAP, SPICE); software evolution (the work of Lehman and Madhavji); software measurement; software project management; and examples of software project experience databases. We discuss and evaluate each of those topics and then define our research directions for this thesis. In chapter 3 we first clarify the terminologies which are used in the rest of the thesis, and then state a set of requirements dividing into three areas of concerns: basic requirement for a PSEE; additional requirements for software project; and requirement for managing software evolution. Those stated requirements are used to direct and evaluate our work reported in the thesis. We then present the EPOS system as a PSEE prototype in chapter 4. A categorization framework for process evolution is then introduced in chapter 6 together with a framework to characterize software projects. EPOS approaches to manage bottom-up process model evolution (i.e. manipulation of task network layout and revision of task schedule/resource) are described in chapter 7. Furthermore, an empirical evolution planning approach to manage unplanned process evolution is also presented. Practical implementation issues in EPOS are described chapter 8. The categorization framework and approaches are implemented in EPOS and necessary tool supports are provided. We conduct a case study with a real software company to record and categorize process evolution from both completed and ongoing projects in chapter 9. The collected evolution data are stored in the EPOS experience database for analysis. The most frequent evolution patterns are then recognized and found basis for suggesting improvement initiatives for the studied company. EPOS operations to handle bottom-up process model evolution are then validated with real evolution data from the case study.
10.3 Main Achievements

In the following, we summarize some important achievements which are presented in this thesis.

EPOS initial implementation We have designed and implemented a PSEE, called EPOS which supports and integration of both configuration management (EPOSDB) and process management (EPOS-PM). The EPOSDB is implemented on a relational database C-ISAM augmenting with versioning management (COV), transaction management and object access (DDL and DML). COV is an original versioning approach associating with external functional changes of product rather than with versions of products in other traditional versioning methods. The transaction model offers long, nested transactions for software activities, and flexible, user-defined protocols and supports for cooperation of work. The EPOS-PM provides an object-oriented, reflective PML, called SPELL to describe production process and meta-process. A set of process tools are implemented to assist model definition (Schema Manager, Task Network Editor), instantiation (Planner), enactment (Process Engine) and evolution (Schema Manager). Some process examples have been modeled and enacted in the context of EPOS-PM to validate our achievement. (refer to the comprehensive evaluation of our work in section 10.4).

Categorization framework We have defined a flexible framework to categorize process evolution to better understand its property and impact. The framework can be configured and adapted to particular situation of the software company. The evolution occurrences, their associated impact are then classified and recorded together with the information about the project in which they occur. Among the recorded project information, there is project profile measures or indicators. Such project profile indicators are associated with a weight and then used to determine similarities between projects.

Approaches to manage process evolution We have elaborated approaches to manage both top-down and bottom-up process model evolution in EPOS. The bottom-up process model evolution approaches are highly inspired by the project management facilities to deal with changes to project plan, e.g. adding or removing activities; revising schedule estimates and adding or removing resources of upcoming activities. In addition, we have also presented an approach to improve project planning practices, especially project estimation, based on the empirical evolution data and analytical information in the EPOS experience database.

EPOS extension We have extended the EPOS baseline to integrate the categorization framework and associated approaches. The existing data structure is augmented to represent evolution data and other relevant project information. A prototype of experience database in EPOS is established to contain histories of completed projects. A set of tools are then implemented to characterize project, record/analyze evolution occurrences and select similar completed projects. Those new tools are working closely with the existing EPOS process tools to provide an integrated PSEE which effectively manage process evolution.

Case study We have then validated our achievement by conducting a case study with a real software company. The categorization framework is thus exploited to record and classify process evolution occurrences from both completed and ongoing projects. Our proposed empirical evolution planning approach is then applied to firstly establish a set of baselines representing useful information of the company such as: estimation capability, effort distribution, pre-release errors and evolution data. Those baselines are associated to project profile indicators in such a manner that
planning practices of new similar projects can be improved. The EPOS experience database and its associated tools have been used to record and derive analytical information for the studied company.

10.4 Evaluation

In this section, a complete and comprehensive evaluation of EPOS and our work is presented first by an overall evaluation followed by an evaluation against the stated requirements in sections 3.6, 3.7 and 3.8. An assessment of our achievement comparing to other existing technologies is then given before we finally conclude with an outline of criticisms or disadvantages or our system.

10.4.1 EPOS Highlights

Below, we give a summary of the highlights of the initial implementation of EPOS and our extended features in managing evolution. More detailed evaluations based on different areas of concerns are presented in the following subsections.

- **EPOS** is an **instrumentable kernel** rather than a full fledged PSEE. Therefore, it allows the flexibility and open-endedness required to accommodate the variation in projects and facilitate integration of new tools. A set of kernel process models is used to instrument the EPOS tasking framework.

- **EPOS integrates both configuration management (CM) and process management (PM).** The CM process can easily be described in PM to guide or enforce the process of change to software artifacts. On the other hand, Process information by means of types is stored as well as versioned similar to any other product data. It means that, EPOS has tightly integrated two discipline in a single environment.

- The EPOSDB adopts **COV as underlying approach for versioning.** COV is an original approach for version and configuration management implemented in an object-oriented software engineering database.

- The **long and nested EPOSDB transaction model** is suitable for software engineering activities which may last for weeks or months. The rather strict change propagation semantics when commit is extended with more flexible support for user-defined cooperation protocol, events/notification about conflicts, easy information access to resolve updating conflicts effectively.

- **EPOSDB offers a simple OOER (object-oriented and entity-relationship) data model with a well-defined DDL and DML.** A small set of predefined types and meta-types in the kernel database facilitates for further customization and refinement. The meta-type is explicitly represented to offer facilities for access and evolution of type information. Both the passive data (products) and active data (task information) are consistently described represented to ease manipulation of them in an uniform manner.

- **EPOS-PM offers a process modeling language (PML) - SPELL which can be characterized as a hybrid modeling approach.** This is due to its combination of existing modeling paradigm
such as: net-oriented (task network), rule-based, active database (trigger), work flow (work decomposition and horizontal chaining). The decision to apply several paradigms is made to optimally exploit their goodness and advantages. In addition, SPELL is a self-reflective modeling language. It means that both processes and meta-processes can be explicitly described by the same language.

- **SPELL extends the underlying DDL/DML of EPOSDB** with i) constructs to represent tasking framework (e.g. PRE-, POST- condition, FORMAL, DECOMPOSITION); ii) more flexible inheritance rules (e.g. redefinition, concatenation, Smalltalk's inner); iii) behavior object-orientation such as procedures and trigger definition; iv) separation of instance- and type-level attributes and procedures; v) high-level language macros to ease the modeling tasks.

- **EPOS-PM offers suitable supports for specifying or modeling processes.** The Schema Manager provides facilities for process definition either in textual and graphical manner. The Task Network Editor gives a more comprehensive and flexible way to construct the task network to reflect the intended process.

- **EPOS-PM applies an rule-based (AI) process planning or reasoning.** In concrete, the EPOS task network is configured and constructed based on process information, goals and product constraints by the Planner. The planning process is carried out automatically and incrementally to detail the task network whenever necessary.

- **EPOS-PM Process Engine Process dynamically interprets and execute the task network.** The tasks are carried out both in sequential and parallel manner. The Process Engine allows automated execution of trivial task which mainly involves tool invocation, and facilitates for interaction in human-performed tasks.

- **The process models in EPOS are consistently evolved to adapt to changes.** The support for model evolution is provided for all levels of abstractions or variation (i.e. generic, enactable and enacting process models). By applying late/dynamic type binding, changes to generic process models are immediately propagated to living instances before an impact analysis is performed incorporating some user decisions (top-down evolution). In addition, the extension of EPOS supports bottom-up evolution, i.e. changes are done to enactable/enacting process models without affecting the generic ones. Changes are encapsulated, locally effective until users explicitly decide to propagate them to the corresponding generic process models.

- **EPOS-PM manages to integrate adequate features from project management** field such as: project initiation, status tracking, effort collection, addition or removal upcoming activities, and revising activity schedule or allocated resources. All those features are operationalized by the Project Manager which tightly work together with other process tools to provide a fully integrated PSEE.

- **A categorization framework is proposed to increase the understanding of unplanned process evolution** during software projects. The categorization framework is used to record evolution occurrences, their consequences together with project characteristics to improve the predictability and anticipation of future similar projects.

- **An empirical evolution planning approach** is operationally proposed to reuse past experiences in future projection of process evolution. It takes into account the experiences from completed
projects to assist in making new project plan. In doing so, the project plan is anticipated with likely evolution impact, and thus better react to changes.

- An experience database is realized to record historical data which are retrieved to support the planning approach. A set of tool have been implemented for recording evolution occurrences, characterizing project, selecting best-matched completed projects and analyzing evolution data.

10.4.2 Evaluation wrt. Objectives

In this section we evaluate our stated motivations and objectives which are stated in sections 1.2 and 3.4 based on our concrete research achievements.

M1/O1. Integrating SCM and project management facilities into PSEE. Our PSEE prototype EPOS offers supports for both software configuration management (SCM) and software process management (PM) in an interconnected manner. EPOS-PM also provides high-level supports for project management such as project initiation, planning, monitoring (collecting project measures) and adjusting according to evolution. Those supports are provided to users in an integrated manner within EPOS environment. A set of coarse-level and project-oriented metrics are defined to characterize project. This metric set can easily be extended to accommodate to particular project needs. However, the project support provided is still at high-level, and thus not comparable to a full-fledged, commercial project management tool. It is therefore space for making further improvement. The rationale of this work is rather to demonstrate the integration possibility of other disciplines into a PSEE.

M2/O3. Managing process evolution within PSEE context A flexible and instrumentable categorization framework is defined to improve understanding of process evolution. By applying such a framework the information related to process evolution is recorded and made available for analysis and reuse in future similar projects. Suitable approaches are defined to handle top-down and bottom-up process evolution, and integrated as support facility for project management. The underlying assumption for effectively managing process evolution with those approaches is to gradually build up experiences and then systematically reuse them in similar circumstances. It therefore requires a commitment for up-front investment in building up experiences.

M3/O4. Validating PSEE in an industrial setting We have conducted two validation phases. The first phase mainly base on the fabricated process examples and a simple industrial review process. The benefits of the validation are two-fold. The gained results firstly validate the applicability and coverage of our modeling concepts and enactment support. Secondly, they direct our further research to improve EPOS. The second validation phase is a full case study with a real industrial setting. The categorization framework is adopted to gain better understanding of process evolution in a sample of projects. The results obtained are analyzed and packaged in EPOS experience database for future usage. The case study provides us some valuable answers to our questions:

- A rather informal process modeling language is best suitable for industrial software processes. It is confirmed by the fact that organization's quality system, serving as standards, process templates, is described by e.g. flow diagrams and prose.
There is no need for computer-assisted process enactment during project execution. Rather there is a need for computer support in generating reports, collecting measures, etc. That is, a hybrid process enactment support is most suitable for industrial processes. The working environment and tasks are too turbulent and dynamic for project personnel to be enforced in a strict and straightforward manner.

There is a need for having a shared experience database with online access across projects in the company. The information in the database must be systematically stored and easily extracted for any purpose. This is a significant means for disseminating experiences, results to all personnel.

In the following, we will evaluate our achievements structuring into the three areas of requirement for a PSEE which are stated in chapter 3: basic requirements, supporting software project, software process evolution.

10.4.3 Evaluation wrt. Basic Requirements for a PSEE

Modeling concepts We have introduced a set of modeling concepts to facilitate modeling phenomena within software processes (section 4.5.2). They include task, product, tool, person, and human-role. Those modeling concepts are consistently related and familiar to any body. On the basis of our experiences gained in modeling different process examples in chapter 5, our concepts seem to satisfactorily meet the requirements on sufficiency, consistency and familiarity. In addition, our process modeling language SPELL (see below) extends the existing modeling concepts with new features to improve its expressiveness.

Process modeling language We have introduced a PML, called SPELL (section 4.5.3). The textual definition of SPELL is still difficult to understand for a novice due to the adopted clumsy Prolog syntax. We have therefore designed and implemented tool support which offer graphical notations, to facilitate the modeling task (e.g. Schema Manager in section 4.5.6.1, Task Network Editor in section 4.5.6.2). However, SPELL provides a solid degree of formality which enables process model analysis. Due to the object-oriented property of SPELL, it provides both abstraction and modularization of the process models SPELL facilitates for extension and incorporation of new concepts. We have defined 6 process models in section 4.5.4. They represent task, product, tool, resource, project and meta-processes. The result of process modeling exercises indicates that our process models are adaptable, extensible and capable in expressing necessary processes. A set of constraints are explicitly stated in 4.5.4.7 to maintain the consistency of our process models. Our process models are modularized to such extent that it is possible to reuse part of them in modeling new processes (see examples in chapter 5).

Process model instantiation EPOS-PM Planner (section 4.5.5.2 provides method and tool support for constructing a task network (or a plan) from process models and existing software artifact. Such a meta-support is performed automatically and incremental. The generated plan is both vertically decomposed and horizontally chained. The task network is statically generated and possibly re-generated when modifying product structure and process models.

Process model enactment The generated task network is interpreted and enacted by EPOS-PM Process Engine (section 4.5.5.1). Currently, the Process Engine must be manually initiated periodically. An improvement are made by Project Manager Tool (section 4.5.6.6) to notify user that
something has been happened, and the Process Engine should be re-executed. By that reason, the Process Engine does not meet the stated requirements on automation and interactivity. The Process Engine allows tasks, ready to be executed, perform in both serial and parallel manner. A cache is used to store invariant part of type information so that the efficiency and thus response time have been considerably improved. Due to strong platform and language dependency (Unix and Prolog), it is not simple to move the Process Engine to another platform.

10.4.4 Evaluation wrt. Requirement for Supporting Software Project

Data management COV 4.4.2 provides an approach for managing different versions of information stored in the EPOSDB. The way in which version and configuration are selected is described in 4.4.3.

Cooperation EPOSDB cooperative transaction model (section 4.4.5 and 4.4.6) provides an overall architecture appropriated for cooperation support. The coordination requirement is provided by Transaction Planning Assistant in section 4.5.6.4. Communication between and within EPOSDB transactions is supported and presented in 4.4.7, 4.5.7.1, and 4.5.7.2. CHAT (section 4.5.6.3) provides flexible support for cooperation at product and workspace level (collaborative).

Organizational aspects Few simple organizational aspects have been modeled. They are presented in sections 4.5.4.4 and 4.5.4.5.

Project management support A project plan or an enactable process model is constructed by the Planner at the first time. The Planner doesn’t take into consideration the project parameters such as available time, cost and resources. However, the enactable process model or task network in EPOS can be manually manipulated by adding or removing tasks and assigning or revising schedule and resource of forthcoming tasks before the project starts (see section 8.4). During project execution, the efforts spent in performing tasks, are automatically collected. The enacting process model (i.e. the EPOS task network being executed) can also be dynamically manipulated to adapt to changes by similar operations which are performed to the task network before start. The initial implementation of Project Manager Tool (section project-manager) in EPOS has been upgraded to integrate those new features which are described in section 8.4. The EPOS data structure is extended to record histories of completed projects for future projection. An experience database in EPOS and associated tools are thus designed and implemented (see section 8.3).

10.4.5 Evaluation wrt. Requirement for Managing Software Evolution

Bottom-up process model evolution As described above, both the enactable and enacting process models are possibly manipulated and revised according to changes during projects. We have both presented the approaches in section 7.5 and implemented them in EPOS (see section 8.4).

Top-down process model evolution A simple procedure for managing type modification is modeled and associated to a meta-type (section 4.5.4.6). This procedure is just a simple example illustrating how a more complex meta-process can be modeled and attached to existing process models, We have also provided tool support for defining and managing process models in Schema Manager (section 4.5.6.1).
Unplanned process evolution We have introduced a framework to categorize process evolution occurring during project (section 6.3). This framework is an effective means to recognize similarities between evolution occurrences as well as their relations to other project characteristics. We have also elaborated an empirical evolution planning approach (section 7.7) to improve both the initial and iterative project planning. The approach is mainly based on the evolution data and the derived analytical information. The categorization framework and approaches are implemented within the EPOS context, and adequate tool supports are provided. Those tools include Project Characterizer Tool, Evolution Registrar Tool, Evolution Analyzer Tool, Project Selector Tool. Such tools are operating on the EPOS experience data to provide support to better manage unplanned process evolution.

10.5 Comparison with Existing Technologies

In chapter 2 - State-of-the-art, we describe our survey of some existing PSEEs, current work relating to software evolution of Lehman and Madhavji, project management field and the Quality Improvement Paradigm (QIP). A summary of evaluations and further work directions are given in sections 2.2.3, 2.4.5 and 2.8. A number of open issues related to existing technologies are identified and presented in section 2.8. Below, we will assess our achievements against those identified open issues.

Process Support Technology - PSEE

We outline following jointly EPOS features which are not satisfactorily addressed by an existing PSEE.

Tight integration of configuration management (CM) and process management (PM) in a single PSEE EPOSDB provides explicit representation of products as well as processes; original versioning paradigm; long, nested and cooperative transaction for software engineering activities. EPOS-PM, residing on top of EPOSDB, offers formalism to describe processes and tools to instantiate, enact and evolve process models in order to guide and enforce software processes.

Flexible means for defining project measures to be collected and analyzed That is, the user defines what project measures to be collected; collection tool (i.e. Project Characterizer) is accordingly configured. Such feature is also provided by some measurement-based PSEEs presented in section 2.5. However, the measurement activity is not incorporated into process models in EPOS to automatically invoke appropriate collection tools.

An embedded experience database in EPOS to record project history, including project characteristics, estimates, actual performance, estimation capability, recognized evolution patterns and evolution data. A set of support tools are implemented to record, retrieve and analyze the historical data for future projection. Of course, it is not our intention to compare our experience database to those at NASA-SEL or SPR with their extensive data foundation over a long period of time.

Operationally support both top-down and bottom-up process model evolution Most existing PSEEs facilitate full support for evolving generic process models, and then immediately or eventually propagating to their living enactable/enacting ones (top-down). In addition, EPOS provides facilities for evolving the enactable/enacting process models without affecting the generic ones.
10.5. Comparison with Existing Technologies

The bottom-up evolution is performed by e.g. adding or removing planned activities, revising planned schedules or allocated resources (cf. re-plan operations from project management tools).

Software Evolution

Section 2.4 presents a survey of current work concerning evolution in software technology. We study Lehman's work on Program Evolution Dynamics and Madhavji's PRISM model of change. Below, we will outline the differences and similarities between them and our achievements.

Program Evolution Dynamics This work focuses on classification of evolving entities in software development and maintenance into classes which have common evolution characteristics. By doing so, common factors and distinctions between classes provide better understanding of the patterns which may be used to direct, control or predict program evolution in similar circumstances. On the contrary, our classification focuses on the change incident itself rather than the evolving item. Lehman's thought of recognizing patterns by classification has however inspired our work to categorize evolution occurrences.

PRISM model of change Our categorization framework and project history are very much similar to the PRISM Change Structure and its change property. However, the number of change properties and their associated values are unlimited and freely and heuristically specified. It therefore implies that such recorded change properties are hard to be formally represented, thoroughly analyzed and reasoned about in the information base. As a consequence, it is difficult to group or recognize change incidents into typical patterns. On the other hand, the model doesn't define any characteristics of the basic items of changes associated to a given change incident. Such relationships are crucial when predicting what kinds of change likely occur based upon a set of external characteristics of a particular item of change. The model also lacks of an operational approach (including tool support) to exploit change-related data to support future projection and decision making process concerning similar changes. We don't have a separate infrastructural structure similar to the PRISM Dependency Structure, to record dependencies between items of change. Such information can easily described in process models. The model infrastructural facilities (i.e. Dependency Structure and Change Structure) are only defined at a methodological level, and thus not properly integrated into a PSEE.

Project Management

We have in our work tried to integrate some high-level project management facilities in EPOS. As discussed earlier, such an integration will imply a clear benefit which is demonstrated in the brief discussion below regarding to identified open issues of project management.

Coarse-grained process follow-up and poor product modeling Integration of high-level project management facilities in EPOS gives possibility for fine-grained process follow-up by enacting explicit process models and automatically collecting project measures. The process model in EPOS is more explicit and fine-grained than the project plan often used in conventional project management tools. EPOS also offers facilities for explicitly modeling product which is not a central support focus in project management tool.
Lack of embedded and global experience database  The embedded experience database in EPOS provides opportunity for recording project data according to pre-defined project metrics and evolution data. The experience database is available across projects with online access. The supporting tools in EDBN provides possibilities for project data recording, report generation and project data analysis.

Poorly managing unpredictable process evolution  We have defined a categorization framework and associated approaches for managing process evolution in EPOS. In concrete, we have elaborated approaches for managing evolution to different variants of process models before and during project execution. The enactable or enacting process models (i.e. project plan) are initially planned with contingency for possible evolution, and continuously adjusted to unpredictable changes in a flexible manner.

Quality Improvement Paradigm and Experience Factory

The discussion of the Quality Improvement Paradigm (QIP) and its experience factory (EF) can be found in section 2.3.3. Our categorization framework and empirical evolution planning approach are highly inspired by the underlying principles of QIP and its EF concerning organizational learning and experience reuse. In our work, we rather narrow our focus on software process evolution and how to reuse such information to effectively plan new future projects. Our primary objective is therefore not to carry out an experiment with a particular technology attempting to achieve a set of goals as in QIP’s steps 2–4. However, QIP’s steps 1, 5 and 6 to characterize projects, analyze and package project experiences are satisfactorily supported by our work (naturally with the focus on process evolution). In addition, we intentionally make our proposed categorization framework and approaches operational and integrate them in a PSEE context.

10.6 Criticisms to EPOS

High degree of complexity  EPOS tries to address and integrate too many relevant research issues simultaneously with a limited resources. The entire system therefore become more and more complex, and hard to maintain due to a high rate of personnel turn-over. The initial solutions or prototypes are rarely followed up and improved further after the involved persons leave. All those works will be a valuable foundation for continuous research in the group if we can restrict or confine our ambition to a realistic level.

Lack of proper documentation  The implementation and main features of EPOS are sometimes poorly documented. Rudimentary manuals or reports are written mainly for internal use, and thus hard to be understood by the external users. Those who are interested, are often suggested to read our published articles which are not intended for novices. New group members also experience EPOS as a high learning threshold, and it therefore takes sometimes before they can properly grasp most of EPOS features.

Lack of external validation  EPOS, similar to most of university research projects, suffer from the lack of external validation in the industry. Without external use of our results, the entire work remains to be immature and is hard to get wide acceptance. Lately, we have managed to initiate
10.7 Further Work

Although a great deal of effort is laid down for our research, there is still considerable room for improvement. In the following, we outline some works which expect to be addressed in the future.

Remaining EPOS Implementation

- We would like to spend more effort in modeling organizational aspects, especially project constraints or parameters such as available schedule, budgetary, resource, etc. Such project parameters should be taken into consideration by the Planner when the original project plan is elaborated. It means that the project plan can be automatically made based on existing process information, product structure, project goal and parameters. Such automatic scheduling should also rely on the past experiences, including actual spent effort, estimation capabilities, process evolusion impact and so on. Rather than making an initial project plan from a scratch, this automatic scheduling will provide a schedule suggestion which is further revised or adjusted by human.

- The EPOS Project Manager Tool offers facilities for manipulating the task network layout and the task scheduling. After performing such manipulating operations, the total task schedule and resource allocation are affected and no longer valid. The Project Manager Tool doesn’t however provide automatic adjustment after such operations. We would like to spend some effort in dealing with such automatic adjustment when the task network layout and scheduling are changed. The infrastructural features are already present such as time attributes associated to task instances, and knowledge about task dependencies in the task network.

- In the current implementation, the categorization framework is constructed and dynamically modified by editing the textual configuration file. Such initial construction and modification in real-time can be supported by e.g. a graphical tool providing facilities for adding, removing and specifying aspect and category information.

- The Evolution Analyzer must be improved to take into account the user-defined criteria to group evolution occurrences to patterns. In the current implementation, the grouping criteria are not explicitly represented at all, but are manually specified by simply ignoring the aspect or category in the framework (i.e. assigning “do not care” values in the Evolution Analyzer Tool). We would like to specify such grouping criteria in a more fine-grained manner, e.g. two evolution occurrences are considered “identical” if they have two specific category values in a particular aspect. The grouping criteria can be formally represented and an automatic recognition facility of evolution patterns from evolution information can then be realized.

- When we performed analysis of the recorded evolution information, the data are retrieved by using the Evolution Analyzer against the EPOS experience database. The data are then exported to another commercial tool to make graphical (e.g. pie chart) presentation. We would like to integrate such presentation facilities into the Evolution Analyzer.
• In our framework and approach, we associate a weight to each of the project profile measure to effectively determine similarities between projects. However, the current implementation (i.e. the Project Selector Tool) has not taken into account such weights when selecting similar completed projects. Such weights are specific for each company and are subjected to change when new insights are gained.

Further Validation and Dissemination

• In our case study, we have not had time to validate our collected baselines in planning a pilot-project. The plan to be carried out has been made and agreed-upon with the project manager at XXX before the planned pilot-project was canceled. It is important to follow up this validation work. An up-starting student work in Spring 97 and possibly the Master thesis as well, has already initiated to pursuit this work.

• In the case study with XXX, we have had a small sample of studied projects. We are not able to perform a thoroughly statistical analysis of the collected evolution data from the case study due to a small number of data points. More data points can be collected from several completed and up-starting projects at XXX. We can then begin to perform statistical analysis to demonstrate significant confidence in our collected data.

• We want to disseminate our ideas and achievements to other software companies which start to establish their own experience databases. We have already assisted the studied software company to establish their local experience database. Our ideas have been adopted to make an experience database on World Wide Web. Still, we need to follow up and expand the existing work to gain more experiences from real industrial cases. In parallel, we would like to make the data transfer between external experience database and EPOS experience database easier. In that way, industry and we can mutually exploit the collected experiences more effectively.

Impact of Future Trends on PSEE

There is a growing interest in workflow technologies, often applied to more automatic office processes [Mar93]; commercial databases; WWW-based development environment with Java as programming language and de-facto standards like Common Object Request Broker Architecture CORBA [omg90], etc... All those new technologies have major influence on the development of PSEE and its architecture in the direction of building federated and distributed systems.

The EPOS architecture is divided in layers which make it easy to replace and then apply such new technologies. Of course, such a decision requires a comprehensive analysis and design. However, we can briefly outline some possibilities as follows:

• The data record layer of EPOSDB can be replaced with a commercial database, e.g. Oracle to increase data interoperability. The version and transaction layers of EPOSDB should possibly do some minor adjustments to operate on the new DBMS platform.

• The EPOSDB is considered as a central experience database, with reusable process models, product models, quality models and project experiences. The object access layer can modified to
be partly web-documents and partly information contained in a full-fledged DBMS (cf. Oracle in the previous point).

- The most abstract level view of the process model should be simple flow diagrams, represented by web-documents with possibility for navigating by means of hypertext or links (e.g. web-based quality system). Other information of process models, such as pre/post condition, scripts, … necessary for formal analysis and computerized enactment, can be annotated or hidden for the users. Web/Java can be used as a bottom-level PML. In that case, the Process Engine must be able to interpret and execute Java-applets on probes model documents.

- A set of tool which is used to instrument, configure, access or manipulate process models, should have a standard user interface, e.g. web browsers. This gives opportunities for standard presentation, query, analysis and synthesis process models upon a shared database.

- The usage of Web/Java for accessing and enacting process models opens a possibility for distributed PSEEs through the network.

### 10.8 Concluding Remark

The initial implementation of EPOS suffers the same fate as other existing research PSEEs when most of the research effort is primarily focused on providing formal supports (e.g. PML) and enactment for process management without getting clearly practical application root in industry. Because of that, PSEEs have not gained much acceptance from industry as expected. We gradually recognize a need for having tight cooperation with software companies to find out what they really need and then gear our research focus towards industrial SPI. The signals that we receive, show a strong indication that software projects suffer with poor planning (including estimation, and scheduling), and that unplanned changes is the primary reason. This has stimulated us to find a better means for better understanding, managing evolution, and trying it out with real cases. We are convinced that our work contributes to gain better control over software development process.

"There is no silver bullet", our research work during this limited period of time and reported achievements are not the exception. This is thus not the end, but more likely a beginning. We have at least contributed to direct focus to the problem of software engineering and process evolution. Categorization framework and approaches are thus proposed and integrated into the EPOS environment with an underlying experience database. Taking into consideration the limited project resources, high rate of personnel turn-over and especially a complex problem to be solved, we are quite pleased with our achievements. Of course, there are still a lots of rooms for improvement which are mentioned in section 10.7. We consider ourselves as both formalists and technologists who struggle in the grey zone between two extremes: i) the ideologists, aiming to develop the "right" way (methodology) to perform processes and ii) the pragmatists, focusing on the study of real-world processes and building systems to support and improve them with little formal support. We therefore ought to find an appropriate balance between those extremes by restricting our ambitions of inventing brand new thing to a feasible level of achievement, and simultaneously seeking for empirical studies to validate the outcomes.
Appendix A

EPOS-PM Manual

A.1 Introduction

In lacking of documented description and a understandable manual of the EPOS system, the author has spent some time in writing this manual facilitating the users to try out the system and consequently being able to model a process by their own. Some of the recently implemented features in the last two year are not included in this manual due to scarce of time. However, the manual is included for the completeness and totality of the thesis. The manual is consisted of two main parts. The first part is an user tutorial where the user is instructed to run some of existing process models to be familiar with the EPOS system. The second part is intended to the users who want to model their processes in EPOS.

A.2 User Tutorial

This section is supposed to give you an overview of user interfaces of our PM-tools and sufficient instructions and guidance on how to start and run two existing process examples. By strictly following the described instructions for running the process examples most of facilities available in this version supporting process enactment are clearly demonstrated. However, if you want to model your processes by using our system we suggest that you refer to section A.3 on Programmer Manual describing technical details on syntax and semantic of our Process Modeling Language.

The section will be organized as following. The requirements for availability of hardware and software are first outlined, followed by the description of how to install and start the system. The primary functionalities are then briefly described by explaining the system's user interfaces. The next two sections are composed of instructions guiding you how to run two process examples: modify_code and ispw6 respectively. Instructions guiding you how to start working from the top-level project (i.e. from organization level), starting sub-projects and so on.

A.2.1 Hardware and Software Requirements

This version is running on Sparc workstation with SunOS 4.1.x which should have more than 10Mb swap memory for each EPOS-PM user to guarantee reasonable performance. Command /etc/pstat -T can be used to check the current available amount of swap memory in your workstation. Unfortunately,
our graphical package (xpce)\(^1\) is not compatible with Solaris yet. The new version of xpce on Solaris is on its way, but not available before the installation and upgrading work are completed. Currently, the only local servers still running SunOS are ask and storlind.

You have to add a new environment variable EPOSDB\_HOST and set it to be ask in your .cshrc, or simply write setenv EPOSDB\_HOST ask at Unix prompt. Your .Xdefaults or .Xresources file must also be added with a set of xpce default values which can be found in file Xdefaults under /home/ask/a/epos/PM-proto/ver1.0. Thereafter, xrdb -merge .Xdefaults must be run to load the new resource values of xpce.

In this version of EPOS-PM the mail program is heavily used to send assignment notifications between team members working within the same project. This program is thus required to be available.

Furthermore, additional requirements for availability of software for each process example will be explicitly stated when necessary.

### A.2.2 Installation and Starting up the System

Please do the following:

1. Create a new directory, e.g. epos under your home directory, and move to it.

2. Run file eposinstall under /home/ask/a/epos/PM-proto/ver1.0 for setting up your system.

3. Following files will then appear under your current directory:
   - EPOSDBclnt: executable file to start an EPOSDB client.
   - epos-gui.pl: Prolog user interface predicates.
   - mailbox.txt and EMmail: facilitating communication between external tools and EPOS-PM's internal execution engine.

4. Run EPOSDBclnt, and get the Prolog prompt (see A.2.3.1).

5. Load Prolog user interface predicates by writing [epos-gui]. Note that each Prolog goal (or command) issued to the prompt must be ended with a dot.

Now you can either run existing process examples (modify_code in sec. A.2.4 or isp_w6 in sec. A.2.5) or start working from the root_proj (see A.2.6 for more detailed instructions). A new beginner is strongly recommended to study the functionality of the graphical user interface described in A.2.3.

### A.2.3 Description of User Interfaces

The primary modeling/enactment features supported in this version will be implicitly presented by describing the functionalities of graphical user interfaces.

There are four main user interface windows when starting EPOS-PM.

1. Prolog prompt

\(^1\)The user manual is available in postscript format under ~-epos/swi/xpce/doc
A.2. User Tutorial

2. Agenda
3. Task Network
4. Mail tool

A.2.3.1 Prolog prompt

This is the window you get when EPOSDBcint is invoked. The prompt starts with 1 ?-. Messages for debugging and execution tracing purposes are displayed here which you don’t have to be worried about. When a graphical window is opened, xpe will due to an internal bug generate a PCE system error message about segmentation violation which can simply be ignored. In that case just type a dot (.) and CR to get back the prompt. Usually, you don’t have to do anything in this Prolog window.

A.2.3.2 Agenda

In agenda window you get everything needed to interact with the process tools. This window is kept open as long as you are connected to the system. Figure A.1 shows how this window looks like.

![Agenda window](image)

Figure A.1: Agenda window

**New Mail:** is a button which will be highlighted when you receive a message (e.g. notification, status-info.) from other EPOS-PM users.

**Status line:** displaying information and guidance for what to do next. Keep an eye on changes in this status line.

**Execute button:** starting to execute the task network by clicking on it which will only invoke a single cycle of execution. When something has been changed (e.g., completion of an activity or an external tool invocation; reception of new status information from sub-projects, ...) this button should be clicked again to enact waiting tasks.

**Schema:** providing functionalities manipulating the schema (i.e. product, process and relation types). It consists of:

- Set editor: setting the schema editor to be either textual or graphical.
- Navigate: start the tool navigating the existing types including facilities for creation of new types, modification and deletion of existing types.
- Edit type: start to edit a given type.
- New type: start to define a new type.
- Delete type: deleting an existing type.
• Load type: create new types expressed in SPELL from a file.

Those functionalities will be explained in more detailed in section A.3.1.

**Workspace:** providing functionalities manipulating with the product structure and mapping (i.e. check-in and check-out) between EPOSDB and workspace\(^2\) in the file system. It consists of:

• Check out: Check out your products from EPOSDB to a file-based workspace. If the current project will develop a new product, and thus no product structure exists in EPOSDB at all, you must do check out anyway to create a new subdirectory (i.e. *myprod*) under the current workspace.

• Product Editor: Allowing you to design your initial product structure with existing product and relation types. It is quite self-explanatory how to use this tool. By selecting Create from this tool the product structure will be installed in both the EPOSDB and file structure will be created in your WS. The validity check is applied interactively during building the product structure preventing from making any illegal construction (invalid relationship between two entities). When a project with already-defined Product Structure the Product Editor can’t be started.

• Check in: Assume that check out has been done and there exists a product structure in EPOSDB. Check in will then copy your products under directory *myprod* back to the EPOSDB, commit the current project and closing all windows. It is important to note that the product structure must be defined (use e.g. the Product Editor described above) before checking in can be taken place.

These functionalities have not been tested properly and thus not available in this version.

**Activities:** displaying tasks (especially deriver tasks invoking external tools) which are ready to be executed. The user is free to start a particular task at appropriate time. Predefined predicates described in A.3.8.5 provide supports for sending activities to the agenda for manual invocation.

**Project:** displaying all sub-projects which have been started from the current project. Their up-to-date project status information will be shown by selecting one of the scroll-down menus. An example of this project status information window is illustrated in figure A.2.

### A.2.3.3 Task Network

It displays the planned task network for you and is illustrated in figure A.3. Remember to click on the scrollbar of the middle view window to get the text inside (again it is a xpce bug).

It is divided in three parts:

1. **Status window:** displaying the status of each product or task displayed in the task network (by selecting *Info* in the popup menu for each graphical node). A *Delegate* button is used to delegate or assign a given task to a particular user (more detail later).

---

\(^2\) file-based workspace facilitating the invocation of conventional tools like cc, emacs, .... It is the directory you stand when you first start the system
Figure A.2: Project Status Information Window

Figure A.3: Task Network window
2. **Log Information**: displaying all status information of the current project and possibly its sub-projects. Information about running, precommit, commit or abort is attached with time duration since the project started.

3. **Task Network**: displaying the task network. A popup menu is attached on each graphical node representing product (ellipse), task or project (rectangle). Below is a brief description:

   - **Product**: *Join* will join two graphical ellipses which are laid on each other. *Split* will on the contrary split one ellipse to two. *Info* displays information about this product to the Status window.
   
   - **Task**: *Info* displays the status of this task to the Status window. *Expand* expand this task to an existing subplan. *Collapse* is the opposite. Note that these two commands can only be invoked to a task when its subplan has been built before. *Re-exec* is not relevant for the time being and thus is grayed out.
   
   - **Project**: *Show Info* shows the status information window as in figure A.2. Other menus are easily to be understood and need more detailed explanation.

### A.2.3.4 Mail tool

The EPOS-PM user may receive a assignment message via email. Therefore you need to start one or another tool to read email (e.g. rmime, xmh, ... in UNIX). This facility of sending message by external mail tool is attempted to be embedded in the system in the next version.

### A.2.4 A simple process example

At the Prolog prompt you run command *start_simple_example*. An agenda and a task network will be displayed. Their functionalities are described in details in sections A.2.3.2 and A.2.3.3 respectively. Please study those sections before proceeding further.

### A.2.4.1 Problem Description and Solution Proposal

This example is simply representing a software development process consisting of editing a source code written in C and compile it. Eventual error message from the compiler will be sent back to the editor for correction. It is decided to model a composite task, called modify_code, encompassing two low-level tasks edit and compile. Products involved are modeled as source_code and object_code while feedback generated by the compiler is called fb_edit. EPOS-PM provides support for delegating work (or a subset/cluster of task network) to several persons locating on different but connected workstations. That is, the real performance of a modify_code, edit and compile can be taken place distributedly. The software requirement for running this process example is availability of emacs text editor and cc tool for C-compiler. This is due to that these tool names, i.e., emacs and cc are embedded and hard-coded in the process descriptions.
A.2.4.2 Running through the example

Assume that there are three persons involved in this scenario. You are the project manager, and you have two software engineers Alice as programmer and Joe as tester in your staff. Your working environment after following steps described in section A.2.2 is illustrated in figure A.4. The directory you stand when you start EPOSDBcint is your workspace under which the real products are stored having the name of the root family which in this case is prod. Due to the fact that the prod directory is a common product space for the entire project, its access rights must therefore be adjusted for all your staff members.

![Figure A.4: Simple process example.](image)

You will now start working by selecting Start working in the popup menu of composite_proj graphical node. A dialog box is appeared asking you to specify the name of the task to be executed. In this example the top-level task is named modify_code as depicted in figure A.5.

Then you are asked to specify your actual products. That is, the input is provided and the output is required when performing this task. Those actual products will be created and stored under directory prod. The types required for input and output are explicitly displayed in the order which the file names must obey when being specified. An example of this is shown in figure A.5. Note that the file name for feedback fb_edit is identical to the file name of source_code. This is because the feedback must statically be designated to be intended to a product. More details about feedback are presented in A.3.8.

By now you will get a modify_code task node connected by actual products in the task network. When clicking on Execute button in the agenda, the high-level modify_code task will be decomposed, and the
whole task network will automatically be generated as shown in figure A.6. Since our graph layout algorithm is not adequately intelligent in locating task nodes, you can always use the middle mouse button to move and place graphical components to suit your need and taste.

Now you should have in front of you an entire process modeled in EPOS-PM for developing and compiling a C source code formalized by a set of tasks connected by actual products. You can then freely choose either to run the entire example by yourself in a single-user environment or to delegate edit and compile tasks to different members in your staff.

A.2.4.3 Single user environment - without delegation

Right after the Execute button is clicked the Execution Manager evaluates PRE-COND and finds out that the condition for invoking task edit is satisfied (task to be executed will be highlighted in the task network). The enactment of edit task causes a text editor (emacs) to be opened in which the C source code is supposed to be edited. After completion the editing task and the text editor is closed, the Execute button should be clicked again in order to trigger a next action. At this time the compile task is qualified to be activated. Usually, most of external tools are purposely modeled in such a way that they can only be manually started by selecting it under the Activities menu in the Agenda. By this way, it limits the degree of automation in which tools are invoked without human control and interaction. The compiler tool cc in this case is first started by selecting its item under Activities scroll-down menu. If the compilation is successfully executed, the enactment of this simple process example is fulfilled. Otherwise, an instance of feedback (fb_edit) containing error message generated by compiler, will be sent back to edit task for corrective action. In the next execution the waiting edit task is waken
up, analyzing the received feedback and acting upon. Because of similar reason above, the feedback content can only be displayed by selecting corresponding item under Activities. Such an iteration will be continuously occurred until a successful compilation is taken place. Remember to write some syntax errors in the C source code in purpose in order to demonstrate this feedback facility. In addition, you are suggested to keep an eye on information displaying in the status field and the New Mail button in the agenda and follow exactly what you are told.

A.2.4.4 Multi user environment - with delegation

Assume that you want to delegate edit task to Joe, having joe as email address. This delegation can be done by clicking on Delegate button in the task network and filling in the information demanded (as shown in figure A.6). The compile task can also be delegated to Alice, having alice as email address, in the similar manner. Joe and Alice will in short time (depending on the traffic load at the moment) receive a message by email notifying about the assignments, and it looks like:

*************** ASSIGNMENT ***************

You have been assigned to work on task: edit
Please run: start_proj(8207, 9633).

*************** ASSIGNMENT ***********************
Joe/Alice will then move to the common workspace and run command `EPOSDBcInt` and load the file [epos-gui] (as described in sec. A.2.2). They will then open their working environments by running the command indicated in the email. From now on they can start executing their assigned tasks within an isolated environment, and precommitting (by selecting `Precommit` in the popup menu of their project task node) their intermediate work to both the parent and sibling project (which in this case are project manager and Joe respectively). That is, the synchronization between `edit` and `compile` is achieved. After precommitting the editing work Joe will keep the same working context, and wait for notifications from his partner Alice about the result of the compilation. Alice is waiting for Joe’s precommit signal displayed in the status bar in the task network before clicking on `Execute` button and executing her process. In the same manner several iterations of `edit/compile` can be performed until the result is satisfied.

After completion of each task a time usage or performance time is calculated and displayed. This time measure is collected by a quality assurance engineer monitoring and ensuring that the current work progress is strictly conformed by the scheduled project plan. This monitoring facility is illustrated in ISPW6 process example.

Since it is just a demonstration your projects should be aborted when you want to leave the system. However, the popup menu for committing/precommitting the project is switched off.

A.2.5 ISPW6 process example

In short, this example aims at demonstrating how a given change request to an existing module is scheduled, performed and delegated among team members. The working progress is continuously monitored and logged. Furthermore, this example also elicit ability for modeling feedbacks. The example also assumes that there already exists a set of products (e.g., change request, module to be modified, project plan, project log, etc...). All those products can be found under directory `prod` under your current workspace.

The textual process specification of this example can be found under `ispw6.txt` under directory `~epos/PM-protolver1.0/examples/ispw6`. To run the example, the steps described in section A.2.2 must be done and followed by command `start_ispw6`.

A top-level task is first created by selecting `Start work` in `composite_proj`, and the actual product structure is specified as depicted in figure A.7.

By clicking on `Execute` button for the first time the `ispw6` task node is decomposed (the first level depicted in figure A.10). The `technical` task can be delegated to another performer who can have a task network as shown in figure A.8.

The entire process with the delegation described above can be seen like in figure A.9, while the real task network for a single user environment is depicted in figure A.10.

By clicking on `Execute` button, and sometimes check in the `Activities` scroll-down menu to start external tools you can keep going until you are satisfied. Due to the similar reason for the simple example above the facility for commit is removed.
Figure A.7: ISPW6: Start work

Figure A.8: ISPW6: Delegation
Figure A.9: ISPW6: Real task network with delegation.
Figure A.10: ISPW6: Real task network without delegation.
A.2.6 Start working from the root_project

This section describes the guidelines for working from the root_project and starting sub-projects manually. Note that the schemas describing the process examples mentioned above aren't available. After have started up the system you do the following:

1. Run then start_epos. and get the EPOS-PM agenda (see in A.2.3.2).
2. Start Root Project under Project scroll-down menu to open a task network (see A.2.3.3) with root project.
3. You are now on your own :-) Good luck!

You will then get the task network like the one shown in figure A.3. You are now playing a role of a manager of a software organization being responsible of initiating goal-driven projects. The current status of projects will be reported back to you and displayed in the Log Information window. Assume that you want now to create a new composite project working with modify_code task (as in sec. A.2.4) assigned to Alice by selecting Create composite project on the popup menu of root_proj. You must then fill in necessary information for this subproject as showing in figure A.11

![Composite Project](image)

Figure A.11: Create composite project

Alice will as a consequence receive an email informing about the assignment which looks like:

******************** ASSIGNMENT ********************

Your supervisor has assigned you to work on a project with name Develop

Change to directory: '/home/rogn/nguyen/epos/test' and run: start_proj(8199).

********************
By changing to that indicated directory and following step 4 and 5 described in section A.2.2 she can now run the Prolog command indicated in the message, namely start_proj(8199). She will then get her own agenda and task network while you are about to receive a message to update running status information from Alice.

Alice will then inspect her goal (by Project Information menu) which is saying Edit and compile file m.c, and finds out that she has a process model supporting this goal. She makes a copy of files ispw6.pl and ispw6-util.pl stored under /home/aska/epos/PM-proto/ver1.0/examples/ispw6 to her working directory. The process model is loaded in by selecting Load types in Schema menu in the agenda (depicted in figure A.12).

![Load Types Figure](image)

Figure A.12: Load Types

She can now start to work by selecting Start working and define the task name, namely modify_code. The utilities must be loaded in by issuing [ispw6-util]. at the Prolog prompt. The remaining is similar to the example in section A.2.4.

### A.3 Programmer Manual

This chapter has a purpose to provide technical details for a process engineer sufficient to define and model his/her processes within the conceptual framework and formalism offered by EPOS-PM. The processes modeled will then be interpreted, planned and executed by the EPOS Planner and Execution Manager as explained in section 4.5.5.2 and 4.5.5.1 respectively. This chapter begins with a presentation of our graphical tool Schema Manager (briefly described in section A.3.1) for modeling software processes. SPELL and its interpreter is then presented. Thereafter, a detailed description of syntax and semantic of type- and instance-level attribute/procedure and trigger is included. Mailbox for communication and feedback modeling is finally described.

#### A.3.1 Schema Manager

Schema Manager is a process tool, which is responsible for generally managing a process schema, i.e. defining, evolving and analyzing a set of types. The Schema Manager tool is attached to the agenda in figure A.1. The popup menu of Schema Manager has a set of menu items. On selecting these menu items, the following events will occur:

**Set Editor:** a dialog box will appear to ask your preferences for the Editor. You can choose between a textual or a graphical editor by simply ticking on the dialog, then pressing OK to accept or
CANCEL to cancel the choice.

**Navigate**: a Schema Navigator (see A.3.2) will be invoked to allow you navigate in the type hierarchies. It gives also possibility in manipulating types (e.g. creation of new types, editing an existing types, ...).

**Edit Type**: a dialog box will appear to ask you the type name. You enter the type name, then press OK button to invoke the editor. The Editor, which contains the type definition, could be either textual or graphical according to your selection.

**New Type**: same as the menu item **Edit Type**, except that the contents in the Editor is a new template definition with the type name you entered as super type.

**Delete Type**: a dialog box will appear asking the type name you want to delete. Press OK to delete it, CANCEL to cancel. Before deletion can carried out, consistent check and impact analysis will be automatically performed to ensure the consistency of the system. Errors will be reported to you, if the deletion will cause inconsistency.

**Load Types**: a dialog box will appear asking the file name of the file that contains type definitions. On pressing the OK, the type definitions will be created.

The defining and modifying of a type is done with the help of type Editor (graphical or textual). Type manipulation, such as creation and changing, is performed in Editor. Consistency check and impact analysis is performed by the Analyst which is embedded in Editor. The type hierarchy of defined types can be browsed by Navigator. Type manipulation, such as deletion or editing, is performed through the menu items in the popup menu of **schema_mgr** task figure, but can be also performed through Navigator (see A.3.2.3). In the following, we will describe the user interface and the functionalities of the Navigator and Editor.

### A.3.2 Schema Navigator

The Schema Navigator allow you to navigate in the type hierarchies. It is started by selecting the menu item **Navigate** in popup menu of **schema_mgr**. The Navigator can display two kinds of type hierarchies by means of trees, i.e. subtype tree and decomposition tree. An example of a subtype tree is shown in figure A.13, and an example of a decomposition tree of task type develop is shown figure A.14.

The subtype trees include three trees: data tree, task tree, and relation tree, with **DataEntity**, **TaskEntity**, **PM_Rel** as root type, respectively. Initially, when the Navigator is started, the data tree is shown up. The tree shown on the screen can be selected through the menu **View**, which contains the menu items **Data Entity**, **Task Entity**, **Relation Type**, and **Find by Name**. **Find by Name** is used to find a specific type by its name.

The decomposition tree allow you to view the decomposition hierarchy of a task type. See A.3.2.2 about how to view the decomposition hierarchy.
A.3.2.1 Navigating in Subtype Trees

The way of navigating is the same for data, task, and relation tree. Initially, only two levels of types are displayed in the tree. To display the subtypes of a lower level type, simply click on the anchor connected to the type (using left mouse-button). If there is no anchor connected to a type, it means the type has no subtypes. If you click on another anchor, the previous expansion will disappear, and the subtypes of the new type will displayed.

To move further up/down in the hierarchy, click the one of the arrows connected to the most left/right type(s), then tree will scroll one level up/down. If there is no arrow(s) connected to the most left/right type(s), the type(s) is a root/leaf(s).

A.3.2.2 Navigating in Decomposition Tree

Through the task tree, you can view the decomposition hierarchy of a task (figure A.14). To do so, click on the desired type (using right mouse-button). A popup menu will appear, which contains four menu items: Edit Type, New Type, Decomposition and Delete Type. Now select Decomposition (other menu items will be explained in next subsection), the decomposition tree with the type as root will be shown up. The way of navigating in the decomposition tree is the same as in subtype trees.

To get back to subtype tree, you can either choose the View menu, or choose the menu item Subtypes in the popup menu when you click on a type with the right mouse button. The former will display the initial subtype tree, and the latter will display a subtype tree with the type you clicked as the root.
A.3.2.3 Manipulating Types Through Navigator

As mentioned above, type manipulation, such as editing, deleting, can also be carried out through the Navigator. Manipulating types through Navigator is much easier than through the popup menu in Task Network, because you don’t need to enter the type name. So if the Navigator already exists or you have a lot of types to manipulate, it is better to manipulate them through Navigator. But if you just want to manipulate only a few types, and don’t want to browse the type hierarchies, the Task Network may be better. To manipulate a type through Navigator, you just click the desired type (using right mouse button), then select one of the following menu items:

**Edit Type:** start an editor containing the type definition of the clicked type.

**New Type:** start an editor containing the new type template with the clicked type as root type.

**Delete Type:** delete a the clicked type.

**A.3.3 Editor**

The Editor is started by selecting the popup menus Edit Type or New Type either in Navigator (A.3.2.3) or in Task Network. Choosing Edit Type will put the Editor in edit mode, while choosing New Type will put the Editor into create mode. There are two types of Editor, i.e. graphical and textual, and you can set your preference through the popup menu Set Editor in Task Network (see the beginning of this section).
A.3.3.1 Graphical Editor

An opened graphical editor is shown in figure A.13. The Editor contains three menus File, Analyze, View and a window showing the graphical representation of the type definition. The graphical representation is composed of several graphical objects: a rectangular body showing type name, left ellipse showing supertype, top and bottom ellipse showing formal-in/out, right ellipse showing the other attributes.

There are popup menus associated to the graphical objects, which are used to add, delete or change the attributes. These popup menus can be showed up by on screen by clicking the graphical objects using the right mouse button. For example, if you click the Formal In ellipse using right mouse button, you will get a popup menu Add New Formal, then a dialog box will appear to ask you some information about the new formal. After you complete dialog, press OK on the dialog box, then a new formal will be created. If you click the small circle representing a formal, you will get a popup menu with two menu item Edit, Delete, to allow you to change the properties of the formal or delete it. The way of using these popup menus is essentially the same.

The menu File: consists of seven menu items:

- Change: make changes to the type definition in EPOSDB according to the content in Editor. The consistency check and impact analysis are automatically invoked before changing. If the changes will cause inconsistency or has some impact on the instances, this information will be reported to you. This menu item is disabled in create mode.

- Create: create a new type definition according to the content in Editor. The consistency check is automatically invoked before creation. This menu item is disabled in edit mode.

- Open: load type definition from a file. Do not work currently.

- Save save type definition to a file. If the file does not exist, a new file will be created. If the file already exists, the new definition will be appended to the end of the file.

- New Editor: open a new Editor window with same mode and content on the same display.

- New Display open a new Editor window with same mode and content on another display specified by display name (consult X window manual about display).

- Close close the Editor window.

The menu Analyze: consists of three menu items:

- Primary Check: check the existence of required-types of this type.

- Extended Check: check if this type is required by other types. Disabled in create mode

- Impact Analysis: analyze the side-effects by changing this type. It does not work currently.

The View menu consists of the following menu items: Type Attributes, Instance Attributes Type Procedures, Instance Procedures, Triggers, Subtypes, Relation Participation. For task entity, there is also Decomposition in addition to above menu items. The View menu is used to change the rightmost displayed properties.
A.3.3.2 Textual Editor

An opened textual editor is shown in figure A.13. The Editor contains two pull-down menus File, Analyze, and an text area showing the textual representation of the type definition.

The menu items in the pull-down menus are the same as those of graphical editor, except that menu item Create is not disabled in Edit mode. That means you can edit a existing type to create a new type. This is especially useful if a new type has little difference to an existing one. But the graphical editor does not offer this functionality, every new type must be created from the given template.

The edit commands in the text buffer are like those of EMACS. If you are not familiar with EMACS edit commands, please refer to EMACS manuals. But we may add some edit menus in the further versions. The textual editor gives you more flexibility than the graphical editor. You can edit any text in the text buffer, but the type definition is easy to be messed up by carelessness. In order to edit the type definition, you must be familiar with the syntax of SPELL (sec. A.3.4). So for the user who is not very familiar with the syntax of SPELL, we recommend using graphical editor.

A.3.4 Modeling with SPELL

This section will give a more detailed presentation of our formalism SPELL providing sufficient facilitates for process designers to model their processes along with the schema manager described above.

SPELL is an extension of Prolog designed for software process modeling based on a structurally object-oriented data model with relation types. Figure A.15 shows predefined types in EPOS which can be customized to accommodate your application.

A.3.4.1 Argument Description

For those Prolog predicates below (+) symbol in front a parameter denotes an mandatory instantiated parameter; (-) specifies variable returned while (?) can be either instantiated or uninstantiated variable. (_) is don’t-care variable.

Attribute list:

```
AL →  X
→  [ ]
→  [NL]

NL →  attr_name-attr_value
NL →  attr_name-attr_value, NL
```

attr_name must be an atom. attr_value may be a variable.

A.3.4.2 SPELL Interpreter

Accessing type-instance-level attributes and invoking procedures expressed in SPELL can be done by an Interpreter, simply implemented by one Prolog predicate.
call_proc(+Caller, +Callee, +ProcName, ?Result/Parameter)
where:

**Caller:** Type name or ID of the caller. It is used to check access right defined in the procedure. For the time being, no check is made at all.

**Callee:** specifying the target the procedure is operated upon. It expects name or ID of the type and instance ID when accessing or invoking type-/instance-level attribute and procedure respectively.

**ProcName:** Name of the procedure. Some predefined procedures consist of `read`, `write`, `read_rel` are used to access both instance- and type-level attributes (see below).

**Result/Parameter:** the value returned after a query or parameter sent to procedure invocation. It has then a format like: `[Attr1-Val1, Attr2-Val2,...]` (simply denote `[AttrList]` from now on).

This predicate supports accessing a particular attributes defined in any supertypes by searching upward in the type hierarchy. Moreover, it also enables dynamic binding of procedures and triggers according to their specified inheritance properties (see below). Some examples below will illustrate the usage of the `call_proc` predicate.

**Querying**

- Accessing the type-level attribute **code** (bound to the variable **Code**) of a task type having ID=300 can be expressed as follows:

  `call_proc(_, 300, read, [code-Code])`

- Querying an instance-level attribute is done similarly but with instance ID instead of type ID.

- Querying the message ID (bound to variable **MsgId**) connected to a mailbox (having ID=200) by relation **contain** and associated boolean attribute `alreadyRead=false` can be specified as follows:

  `call_proc(_, 200, read_rel, [contain, [dst-MsgId], [alreadyRead=false]]);

- Querying an instance **OID** which satisfies some particular instance-level attribute values (i.e. `taskstate=active`) defined in taskentity type can be done by:

  `call_proc(_, OID, read, [taskentity, taskstate-active])`

Another example is to query an ID of a project (ProjId) encapsulated by an EPOSDB transaction with ID=137,

`call_proc(_, ProjId, read, [project, db_trans-137]).`

**Updating**

There are two alternatives:
1. Updating an instance- or type-level attribute value is done by specifying an instance or type ID respectively to the \texttt{call\_proc}. E.g.,
\begin{verbatim}
call\_proc(\_, TaskTypeID, write, [code-NewCode]) or
call\_proc(\_, TaskID, write, [taskstate-NewTaskState])
\end{verbatim}
Note that only one attribute at a time is allowed to be changed using \texttt{write} procedure.

2. Updating an instance-level attribute can also be done by using \texttt{i\_change} procedure defined in \texttt{pm\_rel} (see sec.A.3.5). This procedure can in the contrary update several attributes simultaneously.
\begin{verbatim}
call\_proc(\_, MailboxID, i\_change, [[empty\_true, num\_of\_msg-6]])
\end{verbatim}

\textbf{Constructors}

Similar to a constructor in object-oriented language two procedures are predefined to provide supports for creating instances of a certain type. Those procedures can be invoked by using \texttt{call\_proc} described above. All types have inherited those constructors which can also be extended by new type.

- \texttt{i\_createE(+Attr, -OID)}: create an instance OID of an invoked type having attribute value defined in Attr.
- \texttt{i\_createR(+RelName, +Direction-OID, +Attr)}: create a relationship between an invoked instance to another instance OID having attribute Attr. Direction is either \texttt{dst} or \texttt{src} denoting the direction of OID according to the invoked instance.

E.g. besides creating an instance (\texttt{FooID}), new type \texttt{foo} wants also to attach a mailbox to its instance. In the type definition of \texttt{foo} the constructor will be redefined and will be invoked as follows:
\begin{verbatim}
type\_proc([name-i\_createE(Attr, OID), access-public, inher-inner])
proc\_body(head-i\_createE(Attr, OID), body-(code for attaching a mailbox, ...))
call\_proc(\_, foo, i\_createE, [[name-fooinstance], FooID])
\end{verbatim}

\textbf{A.3.4.3 Domain, Protection and Inheritance}

\textbf{Attribute domain} - domain-

for instance- and type-level attributes. Available domains are: string(N), prologterm(N), short, bool, long, float, char - where N denotes length.

\textbf{Protection} - access-

declared by instance- and type-level procedures. This restriction is similar to private and public constructs in \texttt{cpp} language limiting the visibility and invocation of a particular data member or method. They are defined as:
private: a Caller in call_proc is restricted to this very type.

public: All types and instances can be Caller of the call_proc procedure.

Default value - default-

specifying the default value for a newly created instance of this type.

Inheritance - inher-

for type-level attribute; instance- and type-level procedure to define their inheritance semantic with respect to their supertypes. There are currently three ways to model the inheritance:

redef: overloading the definition of supertype.

append: logical conjunction.

inner: similar to INNER in Simula. E.g. assume that type T defines a procedure p()-a, inner, b. Its subtype TT tries to concatenate the same procedure p() by defining p()-c. By invoking procedure p() on an instance of TT gives the following sequence of execution: a, c, b. Similarly, p()-a, b, inner is a special case of append.

A.3.5 Predefined Schema

This section describes in detail the predefined schema depicted in figure A.15.

TaskEntity and DataEntity type are serving as abstract types representing roots of activity and product types respectively. Activities (or tasks) requiring interaction between tool and human (e.g. design, implement) are usually defined as subtypes of Interactor while tools invoked and executed automatically (e.g. compile, link) are supposed to be modeled as subtypes of Deriver.

Modeling software products is typically done as subtypes of Component distinguishing between Text (e.g. source code, document) which is associated with a Longfield storing the file content in EPOSDB, and Binary (e.g. object code, library) representing derived objects. Every instance of those types has an attribute called contents storing its filename, and status specifying the current status.

Subsystem and its subtype Family define product configuration (inspired by the product configuration in ADELE) connected to its Component by relation Composition having attribute name storing filename of component. New relation types representing coupling between product and its subsystem should be defined as subtypes of Composition. Relation Family of connecting between Family is used to build a hierarchy of product configuration. The product root of the current project is the family root whose hierarchy of sub-families is mapped correspondingly to the directory hierarchy in the file system.

Person and Humanrole can be extended if necessary to describe your organization. But the restriction for performance of certain tasks by certain roles has not fully implemented in this release. A system role being a subtype of humanrole has already been defined to restrict the authority of starting a root_proj. The role aspect will hopefully be improved in the next release.
Task_td and data_td represent meta-types of all activity and product types respectively. Procedures defined in those two meta-types are considered to serve as meta-activities which manipulate and cope with type evolution. At the time being, definition and use of meta-activity is still limited, and will be explored further in my future work.

A graphical editor is used to model your processes (more about them in sec. A.3.1). However, it is still useful to know the textual definition of process definition expressed in SPELL. Task and product definition can be divided into two parts: instance- and type-level. Each part is again composed of attribute and procedure definition. In the following the semantics of each part are tried to explained (mostly properties of task type - properties of data type are subset of them). In Prolog all variables are started with a capital letter while constant value has small letter at first. Filename must be specified as an atom, e.g. ‘myfile.tex’. In the specification of Prolog predicates, parameters prefixed by ? are optional, by + are mandatory input parameters, and by - are output parameters.

A common root for TaskEntity and DataEntity is pm_entity including shared procedures categorized into two classes:

**instance creation:** (similar to constructor in OO) is realized in type-level procedures i_createE and i_createR. They are used with call_proc as below:

```prolog
call_proc(+Caller, +Callee, i_createE, [+AL, -InstID]).
call_proc(+Caller, +Callee, i_createR, [+RelName, +Direction-OID, +AL])
```

*Direction* is either src denoting that OID is source of the created relationship or dst specifying that OID is destination of the relationship.

**instance evolution:** is implemented in instance-level procedure i_change and is used with call_proc
such as:
call_proc(+Caller, +Callee, i_change, [AL]).
When this procedure is applied to a product performing as an input to a waiting task, the taskstate of this task will then be assigned to be \textit{hot}. That is, this task’s conditions will be evaluated immediately.

Note that subtypes of \texttt{pm\_entity}, \texttt{taskentity} or \texttt{dataentity} can of course redefine those three procedures described above.

\section*{A.3.5.1 Product}

The schema of product hierarchy:

\begin{verbatim}
type dataentity is pm_entity;

type part is dataentity {
    prologterm(40) status = created;
    prologterm(40) accessrights = private;
}

type subsystem is part;

type family is subsystem;

type component is part {
    bool primary = true;
    prologterm contents = '';
    float timestamp = 0.0;
}

type text is component;

type binary is component;

type humanrole is dataentity {
    prologterm(40) view = '';
    prologterm(40) rights = '';
    prologterm(40) responsibility = '';
}

type person is dataentity {
    prologterm(16) name = '';
}
\end{verbatim}
A.3.5.2 Activity

type taskentity is pm_entity {
    prologterm(10) taskstate = created;
    short currproj = 0          /* Project ID to which task belongs */
    long starttime = 0          /* Clock time from task turns to be active */
    short assigned_trans = 0    /* Transaction ID to which task is delegated */
}

type deriver is taskentity;

type interactor is taskentity;

A.3.5.3 Relation

reltype pm_rel (pm_entity: 0..n, pm_entity: 0..n);

reltype composition is pm_rel (subsystem: 0..n, part: 0..n) {
    prologterm(40) name = '' /* is usually used to store filename or familyname */
}

reltype familyof is composition (family: 0..n, family: 0..n);

reltype subtask is pm_rel (taskentity: 0..n, taskentity: 0..n) {
    short seqNo = 0       /* storing sequence number of subtasks */
}

reltype actualparms is pm_rel (taskentity: 0..n, dataentity: 0..n) {
    short seqNo = 0;       /* storing sequence number of input/output */
    string(40) paramName = '';/* actual filename */
}

reltype geninputs is actualparms (taskentity: 0..n, dataentity: 0..n);

reltype genoutputs is actualparms (taskentity: 0..n, dataentity: 0..n);

reltype jobbassignment is pm_rel (humanrole: 0..1, interactor: 0..n);

reltype employment is pm_rel (humanrole: 0..n, person: 0..1);

reltype owner is pm_rel (person: 0..1, dataentity: 0..n);
A.3.6 Modeling Methods

This section is meant to provide a complete guidance on how different type- and instance attributes, procedures and triggers can be defined. The presentation is divided into instance- and type-level.

A.3.6.1 Instance-level

specifying attributes, procedures and triggers which are private to instances of this type (e.g., taskstate is a private performance state of a task instance).

Attribute

\[
\text{inst\_attr([name-aname, domain-adomain, default-adefault])}
\]

You can define/add as many instance attributes as you like to a particular type. The semantics of domain and default can be found in sec. A.3.4.3.

Procedure

\[
\text{inst\_proc([head-ahead(parameters), access-aaccess, inher-ainher])} \\
\text{proc\_body(head-ahead(parameters), body-(abody))}
\]

consisting of two parts: an interface and body. The semantics of access and inher can be seen in A.3.4.3. In the body of procedure a predefined symbol $self$ represents the instance identifier.

Trigger on Entity

\[
\text{trigger([proc-Proc, when-When, cond-Cond, action-Action])}
\]

where:

- **Proc**: is a full name of the procedure including its parameters in which the trigger is associated.
- **When**: is either after or before specifying whether the defined action part of the trigger is fired after or before the associated action respectively.
- **Cond**: specifies on which condition of the object must be satisfied if the trigger will be fired.
- **Action**: defining the trigger body.

Trigger on Relation

\[
\text{trigger([proc-Proc, when-When, cond-Cond, action-Action, role-Role])}
\]

where most of parameters are similar to triggers associated on entity.

- **Role**: denoting whether the trigger is applied to the source or destination entity of the relationship at hand.
A.3.6.2 Type-level

Attribute

There are three type-level attributes for dataentity which can be inherited or redefined by its subtypes. They are:

- **legalsuf** (no default value): defining the legal suffix of a product type. E.g. [tex, doc] is legal suffix of a document type.
- **initprim** (true as default): describing whether the type is primary or derived object. Source code is e.g. primary while object code is derived.
- **initvers** (true as default): denoting if the type is an initial version - not being used much.

Type-level attribute of taskentity can be perceived to consist of following parts:

- **static_pre**: (default-[]) has format [[StatusInp1, ...]]. The wanted status for each inputs must be defined. If no input exists, leave this condition as it is.
- **static_post**: (default-[]) has format default-[[StatusOut1, ...]]. The wanted status for each output must be specified.
- **dynamic_pre**: (default-[[], []]) has two parts: temporal and non-temporal. A temporal part might be that a task should not execute before all its predecessors have terminated, or that it only should be compiled at night. A non-temporal part might be that the input of the task must have changed since the last execution of the task. The temporal part is a sequence of Prolog predicates while the non-temporal part may be one of the following predicates:
  - **test_status**: holds if the first input has correct status.
  - **or_test_status**: holds if one of inputs has correct status.
  - **and_test_status**: holds if all inputs have correct statuses.
  - **test_status(Expr)**: testing input status according to the logical combination in Expr. An usage example of this predicate can be test_status([o, [a, 1, 2], [o, 3, 4]]) meaning that the predicate is true if either input 1 and 2 have correct status or either input 3 or 4 has correct status.

If this condition is removed from the definition, the condition defined in the supertype will be applied. The default temporal dynamic_pre for an interactor succeeds if all its preceding tasks have completed and its first input has a satisfied status defined in static_pre. The default dynamic_pre for a deriver is evaluated to true if all its preceding tasks have finished and its input(s) is newer than its output(s). The default non-temporal dynamic_pre for both interactor and deriver is empty.

- **dynamic_post**: leave unspecified for the time being.
- **code**: has three parts Pre-, Main- and Post-Code ( [[], [], []]). High-level task having subtasks the Main-Code empty. The Pre-Code is executed before the Main-Code is invoked, while the Post-Code is responsible for cleaning up or error handler and assigning the proper status for the output. Convenient predicates are:
- `currenttask(-Type, -Task)`: to get TypeId and TaskId of the very particular task (like self). This predicate has to be done at the beginning of each code part.

- `activate_tool(+Type, +Task)`: the task will start an external tool. This predicate prepares this tool, and parameters defined in type_attr executor, and finally send its command to the Activity pop-up menu in the Agenda window.

- `change_output_status(+Task, +ListofNewStatus)`: will change the status of the outputs to NewStatus (ListofNewStatus = [NewStatus, ...]) which should be matched with staticpost defined in the tasktype.

- **formal**: defining the horizontal task network construction. It specifies the allowable data types being input/output of a task type. E.g. the *compile task type* having a single sourcecode and multiple interface as its input, generating an objectcode as output, will define its formal as following:

  \[
  \text{name-formal, ..., default=[[in([sourcecode, sing], [interface, list])], out([[objectcode, sing]])], inher-redef}
  \]

- **Decomposition**: defining the vertical task network breakdown. It specifies the repertoire of valid task types being its subtasks. For a develop task applying water-fall development model may e.g. have its Decomposition defined as follows:

  \[
  \text{name-decomposition, ..., default=[analyse, design, review, implement, test]}
  \]

- **executor**: defines the way in which a tool is invoked to perform a task. This type-level attribute explicitly specifies the name of external tool which will be invoked and the parameter binding of its input and output. Its default value has following format:

  \[
  ..., default=[\text{ToolName, Bindings}], ...
  \]

  Bindings is composed of a set of keyword which have following semantics:

  - `outN` or `inN` (N is a number) means a filename of the n-th output or input respectively.
  - `outs` or `ins` denotes filenames of all outputs and inputs respectively.
  - `newer`: will be replaced by the list of input filenames which are newer than outputs.

  E.g. for a *compile task* above the Executor can be defined as following:

  \[
  \text{name-executor, ..., default=[cc, in1]}
  \]

- **signswitch**: defining the switch which will be used in invocation of tools. E.g. `-c` is a switch for `cc` in *compile task type*.

- **def_strict**: (true/false) specifying if all its preceding tasks must be finished before it can be executed.

- **def_strategy**: (default-goaldirected) the task will be terminated after being executed. Another option is opportunistic setting the task in waiting state, and being executed when the status of its inputs are satisfied.

- **legalrole**: (default-humanrole) specifying which role is supposed to perform this task.
Procedure

\texttt{type_proc([head-ahead\{parameters\}, access-aaccess, inher-ainher])}
\texttt{proc_body\{head-ahead\{parameters\}, body-\{abody\}\}}

Similar to its counterpart instance-level procedure, the type-level procedure has also interface and body parts. But the $\$\texttt{self}$ keyword represent now the type identifier in the procedure body.

A.3.6.3 Predefined predicates for defining task type

This section will give you some helping predicates in order to define a task definition in SPRELL. These predicates described below are used to fill in empty slots in either textual or graphical editor.

Below are some useful helping predicates in defining goals. Several predicates with comments can be found in file \texttt{/home/ask/a/epos/PM-proto/project/pl/taskattr-util.pl}. Anyway, you are supposed to define your own predicates satisfying your needs.

Code

has three parts PRE-, MAIN- and POST-CODE. Task having subtasks leave the main-code empty.

- \texttt{currenttask\{-Type, -Task\}}: to get TypeId and TaskId of the very particular task (like self). This predicate has to be done at the beginning of each code part.

- \texttt{activate_tool\{+Type, +Task\}}: the task will start an external tool. This predicate prepares this tool, and parameters defined in \texttt{type_attr} executor, and finally send its command to the Activity pop-up menu in the Agenda window.

- \texttt{change_output_status\{+Task, +ListofNewStatus\}}: will change the status of the outputs to NewStatus (ListofNewStatus = [NewStatus, ...]) which should be matched with staticpost defined in the tasktype.

\textbf{make_goal_taskname}

A high-level task \texttt{ht}, having a empty main-code, can be decomposed to several subtasks connecting as a sub-network by the Planner. A special instance-level procedure used to issue a goal for the Planner to reason on is called:

\texttt{make_goal_ht\{-[Goal]\}}: issue a Goal specifying the type, filename and wanted status of its output. In the high-level task definition the Goal must be created and have a format like:

\begin{verbatim}
thereis(FamilyIds, OutputFileName, OutputType, OutputStatus)
make_goal_taskname(-[Goal])
\end{verbatim}

\textbf{where} Goal has format like:

\begin{verbatim}
thereis(FamilyIds, OutputFilename, OutputType, OutputWantedStatus)
\end{verbatim}
In the body of this procedure all four parameters above must be found somehow. Following predicates will help you to query those information:

- `get_outputinfo(+Task, -OutputFileName, -OutputType, -OutputId):` returns the filename, type and ID of the first output from a `Task` which in the body is usually defined as $\text{self}$. The two returned values are used to in the `Goal`, while the last one is used in the below predicate to find `FamilyIds`.

- `currentfamily(+OutputID, -FamilyIds):` return the current family IDs of `OutputID`.

- `get_wanted_output_status(+Task, -Status):` get the wanted status of the first output from the `Task` which is usually replaced by $\text{self}$.

- `recursive.ones(+FamilyId, +Status)` and `nonrecursive.ones(+FamilyId, +Status):` for task having family as both input and output (usually a high-level task). The former predicate adds goals with `Status` for all subfamilies of `FamilyId`, while the latter issues goals for all the components of the family `FamilyId`. Note that `collect([], -Goal)` must be called after to unify the final `Goal` which have then e.g. a format like: `[thereis(...), thereis(...), thereis(...)]`.

**subgoals_taskname**

```
subgoals_taskname(+OutputFileName, +FamilyIds, +InputType, +WantedInputStatus - [Subgoals])
```

where `Subgoals` has format like:
```
thereis(FamilyIds, InputFileName, InputType, InputStatus)
```

The Subgoal is invoked for each input of a task, and must be defined in the body of this procedure. It means that we need only to create a `InputFileName` depending on the `OutputFileName` and the function of the task. E.g. for a compile task type with m.o as `OutputFileName` the `InputFileName` should then typically be m.c.

Following predicates will ease the creation of `InputFileName`:

- `last(+FamilyIds, -LastFamily):` provides a last family ID in the list of `FamilyIds`.

- `divide_suffix(+Filename, -Base, -Suffix):` e.g. `divide_suffix(test.exe', B, S)` will bind B to 'test', S to '.exe'.

The Planner starts with a `Goal` issued by `make_goal` procedure and builds the horizontal sub-network by reasoning backward. Each newly created task t., having inputs, must specify a type-level procedure called `subgoals_t`.

```
subgoals_ataskname(FileName, FamilyName, InputType, Status, -[Goal])
```

In the definition of high-level task this procedure will be created and issue a `Goal` for each its input which has a format like:
```
thereis(FamilyName, InputFileName, InputType, Status)
```
A.3.6.4 Querying taskstate

\[
\begin{align*}
\text{ins_list}(\text{ProjId}, \text{Elt}, \text{TaskState}) & \quad \text{insert} \\
\text{del_list}(\text{ProjId}, \text{Elt}, \text{TaskState}) & \quad \text{deletion} \\
\text{qry_list}(\text{ProjId}, \text{Elt}, \text{TaskState}) & \quad \text{query} \\
\text{find_list}(\text{ProjId}, \text{Elt}, \text{-TaskState}) & \quad \text{find the taskstate} \\
\text{change_list}(\text{ProjId}, \text{Elt}, \text{Oldstate}, \text{NewState}) & \quad \text{change the taskstate}
\end{align*}
\]

A.3.6.5 Creation of Product Structure

\[
\begin{align*}
\text{enter_family}(\text{Parent}, \text{FamName}, \text{-FamId}) \\
\text{enter_component}(\text{Parent}, \text{FileName}, \text{+TypeName})
\end{align*}
\]

A.3.7 Mailbox Facilities

Each task instance is attached by a mailbox object facilitating communication between tasks. Basic communication primitives include:

**get_mailboxid:** invoked on the task instance and returns an object ID of its mailbox. E.g.,

\[
\text{call_proc(\_ , 123, get_mailboxid, [MboxId])}.
\]

will bind MboxID to ID of mailbox object attached to task having ID 123.

**test_mailbox:** returns a boolean value indicating if the new message is coming or not. E.g.,

\[
\text{call_proc(\_ , 321, test_mailbox, [NewMessage])}.
\]

NewMessage is true if there are new coming messages to mailbox having ID 321. Otherwise, it is assigned to be false.

**write_M:** write a message object or its subtypes to a particular mailbox. E.g.,

\[
\text{call_proc(\_ , 321, write_M, [456])}.
\]

will add a message with ID 456 to mailbox with ID 321.

**read_M:** read a single message (usually the first one) from the message queue of the mailbox. E.g.,

\[
\text{call_proc(\_ , 321, read_M, [MsgId])}.
\]

will bind MsgId to the first message in the queue.

**read_Ms:** read all messages attached to the mailbox. This method returns a list of message IDs. E.g.,

\[
\text{call_proc(\_ , 321, read_Ms, [MsgList])}.
\]

(\text{\texttt{MsgList}}) is a list of message ID for instance \{12, 34, 56\}.  


A.3.8 Feedback

A.3.8.1 Predefined Types

A feedback type has been predefined and has following instance-level attributes:

- **rcvfile**: denoting the input filename of the task where the feedback is intended to be sent to.
- **content**: the name says it all. It is a string of 100 characters.
- **filename**: specifying the name of the feedback file. In some cases people prefers to send feedback in form of a file.

The feedback has two subtypes predefined:

- **humanfb**: associating to human feedback or comment (e.g., comment of the reviewer or trouble report generated by the testers).
- **toolfb**: is derived automatically by execution an external tool (e.g., compiler generates error messages).

A.3.8.2 Modeling with Feedback

Feedback can be added as input or output to tasks at any arbitrary levels. That is, a feedback is produced after a task completion at one level and is consumed by another task at lower or higher level in the task network. The feedback loop must be modeled statically, i.e., the task receiving a particular feedback must be determined in advance when modeling. This fact may set a constraint on modeling real software processes where the destination of feedback is reasoned and decided at run-time. However, our design of feedback and its facilities can reach far in most of the cases.

Each feedback is required to be sent to a task instance if a feedback type is designated to a specific task type, and there are several instances of the same task type. In another extreme, if a task expects to receive feedbacks produced by different tasks, it should utilize the mailbox facilities in sec. A.3.7 instead.

In the following, the methods using to model feedback are described. For simplicity, a producer is the task which generates the feedback while a consumer is the task which reads the feedback.

A.3.8.3 Transmission Feedback at the same level

Assume that we want to model feedback toolfb generated by a compiler and sent back to a edit task.

In the producer task type there are two issues which must be addressed:

1. the static_post for feedback should be defined to be none. E.g.,

   ```prolog
type_attr([name-staticpost, domain-prologterm(200),
              default-[[none]], inher-redef])
```
2. the type-level procedure toolfb_receiver must be defined. This procedure is used to determine the input filename of task which the feedback will be sent to (i.e., the edit task) and has two parameters. The first parameter is either a family name or a filename of the first output, depending on the task at hand. The designated input filename will be returned in the second parameter. E.g., a compiler having .o file as output generates an error message designated to edit task having .c file as input.

\[
\begin{align*}
type\_proc & ([\text{head-toolfb}\_receiver(\text{OutFile, InFile}), \ \text{access-public}, \\
& \ \text{inher-redef}]), \\
proc\_body & (\text{head-fb}\_tool\_receiver(\text{OutFile, InFile}), \\
& \ \text{body-}(\text{divide}\_suff(\text{OutFile, Base, _}), \\
& \ \text{concat}(\text{Base, }'\.c', \text{InFile}))).
\end{align*}
\]

In the consumer task type following issues must be paid attention:

1. static_pre for feedback is specified to be in appropriate state (e.g., initiated).

\[
\begin{align*}
type\_attr & ([\text{name-staticpre, domain-prologterm(200),} \\
& \ \text{default-}[\text{[initiated]}], \text{inher-redef}])
\end{align*}
\]

2. the type-level procedure subgoal_edit must be defined. The goal produced by this procedure for the input feedback must have format like:

\[
\text{SubGoal =.. [thereis, none, InFile, FbType, none]}
\]

where InFile is bound to the input filename of the receiving task and FbType is the name of feedback type at hand.

A.3.8.4 Transmission Feedback at different levels

This problem is occurred when a feedback is generated and consumed at different levels in the task network. The particular feedback is kept moving up to higher level (i.e., to its parent tasks) until its transmission is taken place at the same level in the task network. The two methods described in previous subsection for producer task are still similar in this case for both the parent and the child task. The only difference is in definition of subgoal in the chile consumer task in following manner:

\[
\text{SubGoal =.. [thereis, none, InFile, FbType, done]}
\]

where done is replaced for none as in previous case. This slightly difference is due to the fact that the particular feedback has already been instantiated in the parent task and thus is unnecessary to be created again.
A.3.8.5 Sending activities to the agenda

To be able to limit the degree of automation (e.g., uncontrolled invocation of tools) the user has a need for manually starting the execution of a particular activity at an appropriate time. Following predefined predicates supporting for this invocation delay can be utilized in the code part of task type description. Those predicates send the activation of a task under the Activities menu in the Agenda.

- Predicate for showing feedback content:
  \[ send\_fb\_to\_menu(+TaskID, +Title, +FeedbackID) \]

- Predicate for showing content of the file when selected:
  \[ send\_file\_to\_menu(+TaskID, +Title, +Filename) \]

- Predicate for associating a Prolog predicate to be executed when selected:
  \[ activate\_pl(+TaskID, +Predicate), \]
  \[ where \ Predicate \ has \ the \ form \ like \ e.g., \]
  \[ message(@prolog, +PredicateName, +Parameter1, Parameter2, ...) \]

A.3.9 Dialog Box Library in XPCE

In order to ease the effort of modeling real software processes and building an appropriate user interface in XPCE, a library of dialog boxes is defined. They are extremely useful when making user interface within human interactive tasks and enabling attaching user-defined Prolog predicates to user actions. The interfaces defined in xpcp are described below:

- \[ confirm\_dialog(+Label, +OkPred, +CancelPred) \]
  displaying a dialog box for confirmation or warning followed by an action or selection. This dialog box provides confirmation and cancel options attaching with two user-defined predicates. E.g. an example of usage:
  \[ confirm\_dialog('Are you sure?', [change\_state, Task, Id], []) \]
  will bind predicate \[ change\_state(Task, Id) \] to Ok button and no predicate is invoked when Cancel button is clicked. Note that variable Task and Id must be bound or instantiated.

![Figure A.16: confirm_dialog](image)

- \[ view\_dialog(+Title, +Filename, +ButtonName, +FreeList, +PredList) \]
  returning a window with a view displaying content of a file having Filename and a dialog with buttons defined in ButtonName performing predicates defined in PredList. A boolean value in FreeList indicates whether the view will be disappeared or remained displaying
when the corresponding button is clicked. E.g.,
view_dialog(‘Project Plan’, ‘dialog.pl’, [‘Inspect’, ‘Close’], [false, true], [[inspect, Task, File], [close, Task]])
will open a window with title Project Plan viewing the content of ‘dialog.pl’ (this file), and
having two buttons Inspect and Close invoking predicates inspect and close respectively.
The former won’t close the window while the latter will due to the Freelist = [false, true].

![Figure A.17: view_dialog](image)

- **button_dialog(+Title, +ButtonName, +FreeList, +PredList)**
  has similar appearance to view_dialog but without the view window. E.g.,
  button_dialog(‘Button Dialog’, [‘B1’, ‘B2’, ‘B3’], [true, false, true], [[actionB1], [actionB2],
  [actionB3]]).

![Figure A.18: button_dialog](image)

- **editor_dialog(+Title, +Filename)**
  providing an editor with two buttons 'Save' and 'Quit'. The value of Filename indicates either
  an existing file or an empty file. In the latter case Filename is specified as "". E.g.,
  editor_dialog("Test", "test.txt") or editor_dialog("Test", "").

- **text_dialog(+Title, +LabelText, +ItemName, +OkPred)**
  displaying a dialog box containing user input fields for retrieving information for each ItemName.
  OkPred is invoked with values from input fields as parameters when OK button is clicked. E.g.,
  text_dialog(‘Project Info’, ‘Fill in information’, [‘Project Name’, ‘Owner’, ‘Duration’],
  [create_proj, Task, Id])
  create_proj(Task, Id, T1?selection, T2?selection, T3?selection) will be invoked. Bound var-
  iables Tx?selection (where x=1,2,3) represent string value retrieved from the three input
  fields.

![Figure A.19: editor_dialog](image)

![Figure A.20: text_dialog/4](image)

- **multi_text_dialog(+Title, +LabelName, +ItemName, +ItemValue, +OkPred)**
  provides similar appearance to text_dialog but having each label for each text field. E.g.,
  multi_text_dialog('Project Info', ['Label1', 'Label2'], ['Item1:', 'Item2:'], ['Value1', 'Value2'], []).

![Figure A.21: multi_text_dialog](image)

- **text_dialog(+Title, +LabelText, +ItemName, +ItemValue, +OkPred)**
  similar to text_dialog/4 except from that the input fields will display predefined values specified in ItemValue. E.g.,
  text_dialog('Project Info', 'Fill in info.', ['Name:', 'Owner:', 'Duration:'], ['ISPW6', 'nguyen', '12 days'], [create_project, Task, Id]).

- **info_dialog(+Title, +ItemName, +ItemValue, +OkPred)**
  has similar appearance to text_dialog but only for displaying information specified in the ItemValue
and one OK button with associated predicate. `ItemValue` can either be value constants, variable, or just empty list. E.g.,

```
info_dialog('Info Dialog', ['Name', 'Account'], ['Minh', '2345'], []).
```

- `info_dialog(+Title, +ItemName, +ItemValue, +Button, +Free, +Pred)`
  similar to `info_dialog` but with flexible number of buttons. E.g.,

```
info_dialog('Info Dialog', ['Name', 'Account'], ['Minh', '2345'], ['B1', 'B2'], [false, true], [[actB1], [actB2]]).
```
Bibliography


[GKS+92] Frode Gregersen, Idar A. Kvernevik, Sigurd Skeide, Tor Rune Skoglund, Frode Stokke, Tor Iver Wilhelmsen, and Weicheng Zhu. BEAtE — Type Browser, Editor and Analyzer for ("til") EPOS PM, November 1992. 394 p. incl. Appendices (Student Project Work).


