Experiences in process modeling and enactment

Geir Magne Høydalsvik

December 19, 1996
# Contents

1 Introduction ................................................. 2  
   1.1 Historical background .................................. 2  
   1.2 Research field ........................................ 3  
   1.3 Related research fields .................................. 4  
   1.4 Motivation ............................................. 6  
   1.5 Research topic .......................................... 7  
   1.6 Research methodology ................................... 8  
   1.7 Research Context .......................................... 9  
   1.8 Thesis Structure ........................................ 10  

2 Theory and systems ........................................... 12  
   2.1 Process technology ...................................... 12  
      2.1.1 Process modeling .................................. 12  
      2.1.2 Process enactment .................................. 13  
      2.1.3 Process user types .................................. 14  
      2.1.4 Technical issues .................................... 15  
   2.2 Overall questions ....................................... 17  
   2.3 Existing systems ......................................... 18  
      2.3.1 PME systems ....................................... 18  
      2.3.2 PSEE systems ....................................... 21  
   2.4 Experiences ............................................... 26  
      2.4.1 Process modeling .................................. 26  
      2.4.2 Process enactment .................................. 31  
   2.5 Preliminary answers to overall questions ............ 32  

3 The studied case ............................................. 37  
   3.1 Case study overview ...................................... 37  
   3.2 Characterizing the organization ......................... 38  
   3.3 The Engineering Handbook ................................ 39  
      3.3.1 The Generic Engineering Handbook ................ 39  
      3.3.2 Engineering Handbook overview .................... 39  
      3.3.3 Process descriptions ............................... 40  
      3.3.4 Instantiation ....................................... 41  
      3.3.5 Maintenance ....................................... 41  
      3.3.6 Use ................................................ 42  
      3.3.7 Feedback ........................................... 42
3.3.8 Problems and goals .................................................. 43
3.4 The configuration management system .................................. 43
  3.4.1 PCMS meta-model ................................................. 44
  3.4.2 Control plan ..................................................... 44
  3.4.3 Product structure ................................................ 45
  3.4.4 Creating and controlling products ................................. 48
3.5 Company usage of PCMS .................................................. 51
  3.5.1 PCMS products .................................................... 51
  3.5.2 The product design structure .................................... 53
  3.5.3 The product document structure ................................ 53
  3.5.4 Design and document structures ................................ 54
  3.5.5 Control plan ..................................................... 55
3.6 Summary ................................................................. 57

4 Modeling company processes ............................................ 59
  4.1 The APEL Environment ............................................. 59
  4.2 The APEL process modeling language ................................. 60
    4.2.1 The activity concept ......................................... 60
    4.2.2 Activity chaining ............................................ 63
    4.2.3 Multiplicity and decomposition ................................ 66
  4.3 The APEL product model ............................................. 68
    4.3.1 Properties .................................................... 68
    4.3.2 Inheritance ................................................... 68
    4.3.3 Aggregation ................................................... 70
    4.3.4 Associations .................................................. 71
  4.4 The APEL role model ................................................ 72
  4.5 Systems development ................................................ 74
    4.5.1 Documents .................................................... 74
    4.5.2 Roles .......................................................... 74
    4.5.3 Activity model ............................................... 74
    4.5.4 Project invocation ........................................... 77
    4.5.5 Analysis ..................................................... 78
    4.5.6 Software development ....................................... 80
    4.5.7 System design ................................................ 81
    4.5.8 Detailed design and implementation ......................... 83
    4.5.9 Integration ................................................... 87
  4.6 Document review .................................................... 89
    4.6.1 Documents .................................................... 89
    4.6.2 Roles .......................................................... 90
    4.6.3 Activity model ............................................... 91
  4.7 Change management .................................................. 96
    4.7.1 Documents .................................................... 96
    4.7.2 Roles .......................................................... 97
    4.7.3 Activity model ............................................... 98
  4.8 Summary ............................................................. 103

5 Modeling experiences .................................................. 104
5.1 Activities ......................................................... 104
  5.1.1 The activity concept ...................................... 104
  5.1.2 Activity chaining .......................................... 107
  5.1.3 Activity decomposition ................................. 112
5.2 Products ........................................................... 115
  5.2.1 Concepts and manifestation ......................... 115
  5.2.2 Product modeling ........................................ 118
  5.2.3 Product lifecycles and revisions .................. 120
  5.2.4 Choosing product abstractions ..................... 123
5.3 Human resources .................................................. 124
  5.3.1 The role concept ......................................... 124
  5.3.2 PCMS roles ................................................ 125
  5.3.3 Roles and agents ........................................ 125
  5.3.4 Roles and activities .................................... 126
  5.3.5 Teams and activities .................................... 128
5.4 The process modeling environment ......................... 129
  5.4.1 Process modeling languages ....................... 129
  5.4.2 The product repository .................................. 129
  5.4.3 Generic and specific models ....................... 129
  5.4.4 The process manual .................................... 130
5.5 Types and instances .............................................. 130
  5.5.1 Type level models ...................................... 132
  5.5.2 Instance level models .................................. 134
5.6 Summary .............................................................. 135

6 Programming company processes ........................................ 138
  6.1 Selecting processes to be automated .................. 138
  6.2 Selecting implementation environments ............... 139
  6.3 PCMS*CTS ....................................................... 140
    6.3.1 Change document ...................................... 140
    6.3.2 Change document relationships .................. 142
    6.3.3 Rules ................................................... 142
    6.3.4 Using change documents ............................ 144
  6.4 Process Weaver ................................................ 145
    6.4.1 Agenda ................................................ 145
    6.4.2 Work Context editor ................................ 145
    6.4.3 Cooperative Procedure editor .................... 148
  6.5 Implementation .................................................. 153
    6.5.1 Implementation context and preparations .......... 153
    6.5.2 The two processes .................................... 154
    6.5.3 The three execution environments ............... 157
  6.6 The six alternatives ............................................ 165
    6.6.1 MANUAL - Document review .......................... 165
    6.6.2 PCMS*CTS - Document review ...................... 167
    6.6.3 Process Weaver - Document review .............. 169
    6.6.4 MANUAL - Change management ..................... 171
    6.6.5 PCMS*CTS - Change management ................... 172
10 Conclusion

10.1 Main results .................................................. 213
  10.1.1 Process modeling ........................................... 213
  10.1.2 Process enactment ......................................... 214
10.2 Related work .................................................. 215
  10.2.1 Process modeling ........................................... 215
  10.2.2 Process enactment ......................................... 215
10.3 Future work .................................................. 216
  10.3.1 Process traceability ....................................... 216
  10.3.2 Soft enactment ............................................ 216
Chapter 1

Introduction

1.1 Historical background

The famous term “software crisis” was originally coined at Garmisch in 1968 [97], this is often regarded as the birth of software engineering. Since then, a number of research groups, conferences, and journals have been dedicated to eliminating the “crisis”. That is, to ensure that software can be delivered with the desired level of quality, within estimated time and budget. The problem has been attacked from many angles, e.g., producing better requirements, development methods, representation languages, and development environments. Software Process Technology is seen as a complementary approach to solving software engineering problems. This thesis divides Software Process Technology into Software Process Modeling and Software Process Enactment, each with different historical roots, namely software engineering methodology and software development environments.

Software engineering methodology

The term *software engineering methodology* is often used about software life-cycles as well as methods and techniques applied at various steps in the life-cycle. The first life-cycle model was published in 1970 (Waterfall [104]), and several other models have followed [36]. Methods usually address one or more life-cycle steps, examples include requirements engineering [35], analysis and design [37, 105], software inspection and testing [29], and software maintenance [30]. Methods usually focus on technical issues such as representation languages and tools. Nevertheless, several authors have addressed organizational, social, and political factors, e.g., organization of work [21], projects [38], people [39], management [47, 63, 111], and economy [17].

Although the above references contain excellent work and deep insight there is a tendency of “one size fits all” and “a narrow focus”. First, there is no single methodology that can be used as-is in every software organization, it needs to be customized and adapted to specific needs. Second, methodologies are used within a context, requiring trade-offs between conflicting goals. These problems are attacked by software process modeling.

The term “software process” is often attributed to Manny Lehman [85] in the early 1980’s.
The term is used to denote all activities performed within an organization in order to develop software. The associated term “software process modeling” is used about the activity of understanding and describing a given software process. The overall goal is to improve the software product. However, software products are made by software processes, and in order to understand why a product is flawed (and how it can be improved) one should study the process as well as the product.

Software development environments

Software development environments is software that helps developers to produce software, the usual goal being to produce “better” software in less time. Numerous equivalent terms have been proposed for such tool environments, the most common being software engineering environment (SEE), computer-assisted software engineering (CASE) environment, and integrated project support environment (IPSE) [22].

The term software engineering environment (SEE) is often used about the total set of software tools that are used by an organization in order to produce software. The term integrated software engineering environment (ISEE) is used if the set of tools are conceived as being integrated. SEE’s can be divided into product-centered and process-centered environments. Simplified, product-centered environments focus on the “thing” to be produced, while process-centered environments focus on how the “thing” is produced. It is common to classify CASE as the former and IPSE as the latter, although CASE is sometimes used as a synonym to SEE [86].

As observed by Snowdon and Warboys [112], the scope of an IPSE is inevitably ill-defined; it is always possible to think of something else that could be provided in an IPSE. However, there are some general characteristics, such as holding information about products and projects, configuration management and version control, and tool integration. The term IPSE has evolved into the term Process-centered Software Engineering Environment (PSEE)\(^1\). A PSEE is an IPSE which has an independent process component, i.e., the process is not embedded in the environment, but can be defined by the organization using it. A process is defined in terms of a process program that can be executed. The program execution is called process enactment to pin-point that the execution is performed by man and machine. However, PSEE’s also differ from IPSE’s in that the term PSEE is used about process programming systems without configuration management and version control, and without requirements for project support (PSEE’s as described in [53]).

12 Research field

This thesis is within the field of Software Process Technology (SPT), an approach to solve software engineering problems. It is difficult to give a precise definition of SPT, particularly one that all involved researchers can agree upon. One way to bypass this problem is to define

\(^1\)Related terminology: Process-based software development environments, and process-driven environments in [34]. Process programming environments in [38]
it as topics discussed in workshops and conferences such as ISPW\textsuperscript{2}, ICSP\textsuperscript{3}, and EWSPT\textsuperscript{4}. However, a more accurate definition is needed, and we define it as follows:

**Definition 1.1 (Software Process Technology)** Methods and tools that aim to support the definition and execution of software development processes within a software development organization. Typical support includes modeling, analysis, enactment, and evolution.

This thesis distinguishes between the definition and execution aspects of Software Process Technology, respectively denoted as Software Process Modeling (SPM) and Software Process Enactment (SPE).

**Definition 1.2 (Software Process Modeling)** The activities performed, and languages and tools applied, in order to create models of software development processes within a software development organization.

With respect to software engineering, the overall goal of Software Process Modeling is to contribute to a well defined, analyzable, measurable, and predictable software development process.

**Definition 1.3 (Software Process Enactment)** The activities performed, and languages and tools applied, in order to create and use process programs which support software development processes within a software development organization.

With respect to software engineering, the overall goal of Software Process Enactment is to ensure that the process is followed, and to ease the work of involved personnel.

### 1.3 Related research fields

SPT is related to many other software engineering research fields and software engineering practices. In particular it is closely related to configuration management, quality assurance, project management, and software process improvement. The relationship to configuration management is one of the main issues in this thesis. The other related technologies are referenced, but not discussed in detail. It is therefore useful to distinguish between these related areas and software process technology. In all cases SPT mainly addresses technical aspects and not organizational and social aspects.

**Project management**

Software engineering development efforts are normally organized as projects. Project management is an established discipline which mainly addresses project processes. Thus, there is a

\textsuperscript{2}International Software Process Workshop, 1984–92 [106, 120, 19, 95, 81, 58, 96, 72].


close connection between project management and software process technology. Roman [103] characterizes projects as follows: *Projects are intended to produce certain specified results at a particular point in time and within an established budget. They cut across organizational lines. They are unique endeavors, not completely repetitions of any previous effort.* Lock [88] states essentially the same thing in the following way: "The principal identifying characteristic of a project is novelty. It is a step into the unknown, fraught with risk and uncertainty".

SPT can either be seen as complementary, or competing project management technology. Complementary because, at large, it does not address project issues. From the complementary viewpoint project management tools and methods can be applied in addition to process technology. Competing because there is an implicit assumption among SPT researchers that process technology is a better approach. From the competing viewpoint, process technology will encompass or replace project management technology.

**Software quality assurance**

Software quality assurance is a term that has different interpretations, from the most narrow to the most inclusive, e.g. [44, 54, 70, 77, 99, 121]. When SQA is referenced in this thesis the term is used in its most narrow sense; to ensure that an organization “say what they do, do what they say”. This is almost identical to SQA as advocated by the Capability Maturity Model (Humphrey [64]). The difference is that I do not require that SQA “ensure full compliance”, only that “inadequacies are brought to management attention”. My use of the term SQA resembles ISO-9001 [71, 51] *internal quality audits*, while “process description” resembles the ISO-9001 *quality plan*.

SQA personnel need a description of the software development process, and to observe real-world development processes. On this basis they must decide whether the real-world processes were performed as they were described. SPT addresses process descriptions by process modeling, the real-world process by process enactment, and process conformance by enforcement (process enactment).

**Software process improvement**

Software process improvement (SPI) can be exemplified by the work with the Capability Maturity Model (CMM) [18, 65, 100], the Quality Improvement Paradigm (QIP) [10, 11, 12, 13], and FURPS [55, 56]. When SPI is referenced in this thesis, the term resembles the activities of the ISO-9001 Quality Improvement Team as suggested by Ince in [70]. With respect to CMM [42, 66] it resembles the measurements activities which take place when the organization has reached maturity level 4 (managed) or 5 (optimizing).

SPI personnel need to analyze and evaluate old processes, to describe improvement criteria, and describe new processes. In particular they must describe the data that must be collected in order to evaluate whether the improvement criteria are met or not. SPT addresses “process improvement” in an indirect way by providing process models that can be analyzed and reasoned upon. Furthermore, process enactment can be used for data gathering purposes.
1.4 Motivation

In recent years, there has been a massive research effort in software process modeling and enactment. But the results have not been impressive. A decade after Osterweil’s paper “Software processes are software too” [98] it is still hard to see any substantial impact on industrial practice from SPT. In 1991, Madhavji stated “Although all (…)\(^5\) formalisms attempt to represent process models, it is as yet too early to know how they will fare in the development and evolution of industrial scale processes.” [90], and unfortunately, the same can be said today.

A coarse overview of what has been done in Software Process Technology is as follows (for nuances see Chapter 2):

1. **Theory.** Definition of languages for software process modeling and enactment.

2. **Systems.** Implementation of research prototypes for software process modeling and enactment.

3. **Validation.** Formalization of existing process descriptions found in method handbooks used in organizations.

What has been done should be contrasted with what has *not* been done, revealing a huge need for SPT assessment:

1. **Theory.** There is no well established SPT theory. Theory exists as fragments in published papers, but there is no fragment agreed upon by most researchers. In particular, there is a lack of theory on:
   - The use of SPT in industrial software development. E.g., user needs and SPT applicability (technically, economically, and socially).
   - The coupling between SPT and related efforts, e.g., configuration management, project management, quality assurance, and process improvement.

2. **Systems.** There are very few commercial environments, and existing ones are not used in software engineering, a tendency has been to market such systems as work-flow systems.

3. **Validation.** There have been very few attempts to validate SPT, and existing attempts are unsatisfactory in many respects:
   - Published models tend to be toy examples, not models of processes that actually take place in industrial software development. In 1990, Balzer stated “As a community, we have had far too little experience in actually working with real live processes and the changes they undergo.” [6], and unfortunately, the same can be said today.

\(^5\)He gives examples in the following categories: Process programming languages, Design and analysis paradigm, Behavioral approach, Method-defining schema, Rules, Plan, ETVX, Functional approach, and Knowledge based approach.
To the extent that industrial processes are used, it is the process handbook description which has been modeled, not the work as it is really performed. Furthermore, it is seldom investigated whether the description corresponds to the activities described.

Validations are not performed as controlled experiments, and assessments are not based on real process data.

Modeling is done by researchers, not by process owners. And, to the extent that models are validated by persons in the organization, these are usually not the process performers themselves, but research personnel within the organization, thus lacking detailed process insight.

Validation of languages and tool is usually done by the inventors themselves. Thus, even the few experiences that have been reported are likely to be biased in favor of SPT.

The above list shows a lack of knowledge about industry needs with respect to SPT and the practical applicability of current SPT. There is a huge need for research on these issues, based on sound methods and experiences rather than speculation.

1.5 Research topic

The “topical concern” of this thesis is the applicability of SPT in a software development organization which can be characterized as being industrial, project oriented, and applying a software configuration management (SCM) system.

Within the context of an industrial software engineering project, the research topic is to investigate the technical and economical feasibility of SPT. Above, I identified that applicability have received little attention by researchers. One of the reasons for this might be that it is difficult. It is difficult because the problem involves technical, organizational, economical, and social factors. Furthermore, because these factors must be investigated in the light of user needs. Although difficult, research on the applicability of SPT is essential to understand why the technology has not reached industry, to understand if it will reach industry, and to guide further research.

The problem of SPT applicability is complex and is not likely to be completely answered within a single thesis. This indicates that one should have a narrow focus, investigating only a small fragment of SPT. On the other hand, by choosing a narrow focus one is missing the overall picture. And, since there are no theory on this topic, it is impossible to theoretically relate the results from a focused study to the overall problem.

This thesis combines detailed observations, reported within the bulk of the work, with a discussion of overall questions. Detailed observations must be read as a “story” telling the reader what was experienced. The overall questions is an approach to focus on important issues, questions that can be discussed in the light of the work, but where no definitive answer can be given. These overall questions are stated in Chapter 2 within the following framework:

- **SPT - real world process.** Investigate the relationship between SPT and the real world process. The use of the SCM system is here seen as a part of the industrial process. The
main concern is to what extent SPT is useful, e.g., strong and weak aspects, potential benefits and costs.

- **SPM - SPE.** Investigate the relationship between software process modeling and software process enactment. The main concern is whether it is useful to distinguish between modeling and enactment or not.

- **SPT - SCM.** Investigate the relationship between SPT and SCM. The SCM system is here seen as a tool. The main concern is whether SPT and SCM must be integrated or not.

My primary interest in this work is to gain and report experiences with respect to SPT. The goal is to gain knowledge about industrial needs and the practical applicability of current SPT. This knowledge should be gained by sound methods and experience rather than speculation.

### 1.6 Research methodology

In order to gain experiences I have applied two research methodologies, namely the *case study* and the *experiment*.

The *case study* [116] can be characterized as a study of the specific, what can specifically be learned from the single case. The method applied is to define a “topic of concern”, identify the “case” to be studied, investigate how the “topic of concern” relates to the specific “case”, and on this background make assertions about the “topic of concern”. Rigorously performed, one can state hypotheses about the “topic of concern”, one can design the investigation, gather data in a structured way, and on the this background evaluate the hypotheses.

The *experiment* [82] can be characterized as an effort to generalize from a larger set of observations by the application of statistical techniques. The “topic of concern” is characterized by explanatory variables, some of which are manipulated (predictors) and some of which are measured (predictands). Other, extraneous sources of variation are disturbing variables which are attempted to be controlled or randomized through the experimental design. Experiments must be formulated it terms of hypotheses, experimental design, measurements, and an evaluation of the hypotheses in terms of statistical techniques.

The major distinguishing factor between experiments and case studies is that experiments are repeated in several runs in a controlled environment. The environment is controlled by randomization of disturbing variables as part of the experimental design.

#### The case study

The case study reported in this thesis is a *qualitative* case study [116]. I study software development processes within an industrial software development organization to learn something about software process technology. My *case* is:

- The process descriptions developed and used by one group within the organization,
• the information found in the applied configuration management system, and

• the software development processes as seen by two group members, one responsible for process modeling, quality assurance, and software process improvement, the other being a process performer.

The case study is an instrumental case study, where “a particular case is examined to provide insight into an issue or refinement of theory. The case is of secondary interest; it plays a supportive role, facilitating our understanding of something else.” Stake,[116]. My overall topical concern is the applicability of software process technology. The study is performed to provide insight in process modeling and process enactment, the relationship between them, and their relationship to configuration management. I am not intrinsically interested in the organization's processes, only to the extent that they can bring insight in the use of software process technology.

The overall topical concern is decomposed and formulated as a set of overall questions in Chapter 2. These questions represent the issues to be dealt with in this thesis. I then apply a process modeling language and environment on the identified case. During this activity I make observations related to the issues, these are reported in Chapter 5 (modeling) and Chapter 8 (enactment). These observations are used in Chapter 9, where the overall questions are discussed and answered. The answered questions are refinements to process modeling theory.

The experiment

The experiment reported in Chapter 7 is based on the implementation of two processes, namely change management and document review, identified in the case study part of the thesis. The two processes are realized in two different process support environments, namely PCMS*CTS and Process Weaver. Furthermore, the two processes are performed “manually”, denoted as Manual PCMS, to create a basis for comparison. Data about the execution of process instances are collected from a software project within the studied company. Six document review processes and seven change management processes are selected to be repeated within the experimental context. Both the six review and the seven change processes are performed in Manual PCMS, PCMS*CTS, and Process Weaver. The time used on process steps is measured. (The hypotheses, experimental design, results, and evaluation is reported in Chapter 7).

1.7 Research Context

To satisfy my goal about studying industrial software development processes I contacted various research institutions and software development organizations in Norway and asked for possible “cases”. The result of this search was an established contact with the PERFECT project, through the SINTEF research institute in Trondheim, Norway. Thus, this work is performed within the context of the PERFECT project.

6Process Enhancement for Reduction of software defects, ESPRIT 9090
The PERFECT project is a research consortium consisting of academic and industrial partners in the field of software process improvement. Two academic partners are CAP Gemini Innovation and LGI, both located in Grenoble, France. These two academic partners play the role of tool developers in the PERFECT project. Prior to the PERFECT project CAP Gemini Innovation had developed the Process Weaver process enactment environment, and LGI had developed the ADELE configuration management system. In the context of the PERFECT project these two companies developed the APEL environment together. The goal of the APEL environment is to encompass process modeling, process enactment, and configuration management.

One of the industrial partners is the company in which my case is found (denoted as the Company). SINTEF is a sub-contractor of the studied company, supporting the company in its PERFECT responsibilities. Formally, my own role in the PERFECT project was a SINTEF sub-contractor role. I performed two tasks in the PERFECT project: First, I evaluated the APEL process modeling language and environment with respect to Company process modeling needs. Second, I evaluated the use of Process Weaver with respect to the possible automation of two of the Company processes. At the point in time when my work was performed, the APEL environment was a pure process modeling environment (APEL Version 2). Thus, this thesis discusses APEL as a process modeling language and environment.

1.8 Thesis Structure

The remaining parts of the thesis are organized in the following way:

Chapter 2 (Theory and systems) identifies theory, systems, and experiences in software process modeling and enactment. The research questions are stated in section 2.2, page 17, these questions represent the main issues which are investigated in the thesis. At the end of the Chapter the overall questions are assessed in terms of preliminary answers, answers given by existing theory, systems, and experiences.

Modeling part: The chapters 3–5 contain the process modeling part of this thesis. The case is the software development processes in the Company. These processes are identified in terms of the process handbook, product information in the applied configuration management system, and company personnel. This part presents the studied case, describes the language and developed models, and reports modeling experiences.

Chapter 3 (The studied case) presents the Company and the case. The chapter describes the existing software development environment and software development practices in terms of the process handbook and the configuration management system. This chapter forms the basis and context for the process modeling and process programming activities reported in this thesis.

Chapter 4 (Modeling company processes) presents the applied process modeling language and the developed process models. The process modeling is based on the process handbook of a particular software project, but is supplemented with actual product information found in the configuration management system and information from project
personnel. The process modeling language is interpreted in terms of repository operations. The developed models are depicted and described in this chapter, the modeling experiences are discussed in Chapter 5.

**Chapter 5** (Modeling experiences) presents experiences obtained by the modeling of “case” processes. The chapter can be seen as discussing the relationship between Chapter 3 and Chapter 4. The chapter discusses the use of modeling concepts in the applied process modeling language, how the modeling concepts relate to real-world processes, and how modeling concepts relate to the configuration management system.

**Enactment part:** The chapters 6–8 contain the process enactment part of this thesis. A controlled experiment is performed on the basis of two processes identified in the case study. The two processes are realized in two different process support environments. Furthermore, the two processes are performed “manually” to create a basis for comparison. This part presents the process implementations, the performed experiment, and reports enactment experiences.

**Chapter 6** (Programming company processes) presents the two process support environments and the respective process implementations. The chapter also describes the existing, non-supported execution alternative. Finally, the six execution alternatives, as they are performed in the experiment, are described.

**Chapter 7** (The experiment) presents the experiment in terms of hypotheses, experimental design, data gathering, and evaluation of hypotheses.

**Chapter 8** (Enactment experiences) presents experiences obtained by the process implementation and the process execution. The two support environments are compared and strengths and weaknesses identified.

**Chapter 9** (Evaluating overall questions) presents the answers to the overall questions. The answers represent refinements to existing process modeling theory.

**Chapter 10** (Conclusion) presents the main result, summarizes related work, and a vision for the future.
Chapter 2

Theory and systems

This chapter gives a survey of “state of the art” in Software Process Technology (SPT). Software Process Technology encompasses process modeling methodology, process modeling languages, modeling support, and enactment support. The goals of this chapter are: to describe research issues, terminology, and overall questions; to survey existing modeling languages, existing support environments, and reported experiences; and to give preliminary answers to my overall questions.

2.1 Process technology

This section divide process technology into process modeling technology and process enactment technology. Furthermore, we define terminology related to support and support environments.

2.1.1 Process modeling

![Diagram of processes and their models]

Figure 2.1: Processes and their models.

In software process modeling the real-world objects are software processes. Figure 2.1 depicts two processes (P1 and P2), processes are seen as fuzzy (dotted lines) because it is not possible to observe and understand every process detail, and also because processes are abstractions on human behavior. Process modeling is the act of creating tangible representations (P1’
and P2') of real-world processes. Process models must be created by humans, denoted as process engineers, supported by appropriate methods and tools. Thus, modeling support can be defined as follows:

**Definition 2.1 (Modeling support)** Modeling support is the total set of support given to process engineers for the purpose of process elicitation, description, and maintenance.

An organization may perform process modeling without any tool support, and it may have process modeling support without having automated execution support.

**Definition 2.2 (PML)** A Software\(^1\) Process Modeling Language is a language created for the purpose of describing real-world processes. A PML should have formally defined syntax and semantics.

**Definition 2.3 (PME)** A Process Modeling Environment is a set of computer applications that support the specification and maintenance of process models according to some PML.

### 2.1.2 Process enactment

Process enactment requires process modeling support to implement type-level models as described above. Figure 2.2 depicts the additional process instantiation and enactment support. There are two main differences with respect to Figure 2.1, the instance-level has become explicit and the real-world processes are depicted as human-computer interactions. The instance-level is made explicit to emphasize that it is different from the type-level, and that instances need to be synchronized with the real-world. The human-computer interaction illustrates that process enactment directly interferes with humans, and not indirectly through other people or passive descriptions.

![Figure 2.2: Processes and their models.](image)

With respect to software configuration management (SCM) systems, the major contribution of PSEEs is to add a process concept, an abstraction that is placed between users and SCM

\(^1\)The prefix software may be omitted in the following.
tools, i.e., events from the environment can be directed to processes rather than individuals (represented by tool invocations). This adds a possibility to execute intermediate code between the generated event and its manifestation to the user. This code is written in the context of a process, i.e., it implements desired actions based on process knowledge.

**Definition 2.4 (Instantiation support)** Instantiation support takes type-level process models as input and creates enactable, instance-level models.

**Definition 2.5 (Enactment support)** Enactment support takes instance-level models as input, and based on this information it actively takes part in real-world processes by means of stimuli and responses.

Instantiation and enactment support do not change the process model definition, rather they instantiate, monitor and control process instances. As time goes by, new models will be created by modeling support, also taking into consideration feedback from process performers. Automated execution support requires a process program\(^2\). As of today the process modeling community use “process models” about both enactable and non-enactable models (e.g.\(^8\)). This work separates between the two and therefore use the term “process programs” to denote specifications that can be enacted and “process models” about specifications that cannot necessarily be enacted\(^3\).

**Definition 2.6 (PPL)** A Process Programming Language is a PML. A PPL must have operational semantics and be enactable.

**Definition 2.7 (PE)** A Process Engine is a set of computer applications capable of enacting a process program written in some PPL.

**Definition 2.8 (PSEE)** A Process-centered Software Engineering Environment is a SEE that (at least) includes a Process Programming Language (PPL), a Process Modeling Environment (PME), and a Process Engine (PE). The PME must support people in the act of producing and maintaining the PPL specification, and the PE must be capable of enacting the PPL specification.

According to these definitions every PSEE will also be a PME, but not vice versa. Therefore, the term PME is used about environments that are not PSEE’s.

### 2.1.3 Process user types

Madhavji [90] classifies process users into process engineers, process managers, and process performers. In the following I introduce the main characteristics of these user types and relate them to both process modeling, instantiation, and enactment.

\(^2\) A process model that is given operational semantics.

\(^3\) The difference between process modeling and process programming is similar to the difference between conceptual modeling and implementation in information systems engineering.
• **Process engineers.** Process engineers\(^4\) need to create and change process models. Resulting models should be usable and reusable.

  The process engineer is responsible of process modeling. Process modeling encompasses definition, reuse, and static change (Static change can be defined as change outside the scope of enactment, as opposed to dynamic change that may occur during enactment).

  When developing an enactable model, a process engineer must address at least three extra problems, i.e., *tool integration, work environments, and dynamic model change*.

• **Process managers.** Process managers are responsible for the operational software process. They are responsible for planning, i.e., the instantiation of generic process models into specific process models. Also, they are responsible for initiating, monitoring, controlling, and measuring processes. Process managers have the overall responsibility for process instantiation.

• **Process performers.** Process performers\(^5\) should follow the process model, and give feedback on its usability. Process performers need available, understandable, and up-to-date models. Process performers are those influenced by process enactment. Process performers instantiate processes within the boundaries set by process managers.

Process engineers, process managers, and process performers have different needs which can be supported by software process technology.

## 2.1.4 Technical issues

Here I introduce a set of technical issues, helping the discussion of languages and systems presented in the next sections. Note that these issues will not be used as a classification scheme, which is outside of the scope of this thesis. I divide technical issues into language issues and support environment issues.

### Languages

Here I introduce a set of language issues, helping the description of PML’s and PPL’s which are embedded in surveyed PME’s and PSEE’s.

• **Concepts.** Criteria for languages include scope, relevance of constructs, economy of constructs, understandability, ease of use, degree of formality. The most useful descriptions are comprehensive in scope yet concise in presentation and easy to use. Process modeling concepts should cover several aspects of the software process, i.e., processes, products, resources, and metrics.

• **Abstraction.** Multiple levels of abstraction facilitate human understanding by reducing the amount of detail and by focusing on process context. Abstraction is one of the

\(^4\)I will also use the term “process modeler” in a process modeling context and “process programmer” in process enactment context.

\(^5\)I will also use the term “process agent” as a synonym.
basic skills in engineering and science, needed both when developing models and later for presenting them to others.

- **Modularity.** A modular language facilitates the development of user defined abstractions and reuse by composition.

- **Formality.** Formal languages facilitate a precise process definition, increasing the likelihood of a consistent interpretation by different process agents, at different points in time.

- **Representation.** The most prominent alternatives are text, diagrams, or a combination of these. Diagrammatic representations facilitate human understanding by reducing the cognitive load. Benefits such as faster language understanding and more immediate understanding of models could be expected.

Furthermore, formally defined syntax and semantics is a prerequisite for tool support, e.g., syntactical and semantical analysis, viewpoints and abstractions of large models, validation capabilities, etc.

**Environments**

Here I introduce a set of issues that I will use to write about software process environments (both PME’s and PSEE’s) that will be presented in the next sections.

- **Viewing capabilities** Multiple, complementary views facilitate human understanding by reducing the amount of information presented when a specific aspect of the process is to be understood. Views can be seen as a filter applied to the total amount of information that constitutes the process model.

- **Abstraction capabilities** Hierarchical decomposition could be offered in the various views. Various approaches to constructing abstraction levels such as top-down, middle-out, and bottom-up, could be supported.

- **Verification capabilities** support model conformance to syntactical and semantical rules of the environment, e.g. consistency between views (sub-models). Verification capabilities include consistency, completeness, and correctness checks.

- **Validation capabilities** support model correctness with respect to its intention, i.e., that it represents an adequate and useful abstraction of the modeled subject. Validation capabilities include explanation, animation, and simulation.

- **Repository capabilities** represent the interface to the world of models. This includes model storage and retrieval, sharing models between users, and retrieval of information about models.

- **Project planning capabilities.** The ability to make a model that represents a desirable future and how to achieve it.
• **Monitoring capabilities.** The ability to make real-world observations, in this context through a model of the real-world. Monitoring capabilities include data gathering and analysis.

• **Estimation capabilities.** The ability to calculate interesting prognoses about the future. This requires a model, data, and analysis. The analysis can be deterministic or stochastic.

• **Evolution capabilities.** The ability to change models described by a fixed PML.

• **Cooperation capabilities** Cooperation between concurrent and possibly distributed agents has three aspects: collaboration, communication, and coordination.

### 2.2 Overall questions

The general goal of this work is to investigate both the applicability of software process technology (both modeling and enactment) to real world processes and the relationships between software process technology and configuration management.

The method used is to apply Software Process Technology to industrial software development processes and report problems with respect to applicability. However, I also want to raise overall question that might easily drown in the more detailed observations. These questions are raised in this section, they are furthermore answered preliminary at the end of this Chapter, and finally in Chapter 9.

**Process modeling – real-world process**

- **Q1:** Can existing modeling support (PML and PME) be used for real world processes?
- **Q2:** Which are the benefits of using existing PML’s and PME’s?
- **Q3:** Is software process modeling cost effective?

**Process enactment – real-world process**

- **Q4:** Can existing enactment support (PPL and PE) be used for real world processes (technical feasibility)?
- **Q5:** Which are the benefits of using existing PPL’s and PE’s?
- **Q6:** Is process enactment cost effective?
- **Q7:** Can existing instantiation support be used for real world processes?
Process modeling – process enactment

- Q8: Can every modeled process be enacted?
- Q9: Can every programmed process be enacted?
- Q10: Should the same language and environment be used for both modeling and enactment?

Process modeling – SCM

- Q11: Which are the relationships, if any, between process modeling and SCM?

Process enactment – SCM

- Q12: Can PSEE’s replace SCM systems?
- Q13: Can PSEE’s be used together with any SCM system?
- Q14: If not, what are the requirements for an integrated system?

2.3 Existing systems

In the following I present a selection of PME’s and PSEE’s which I judge to be representative for process modeling and enactment environments\(^6\).

2.3.1 PME systems

This section describes some existing PML’s and their associated PME’s. I have included the SPADE environment that is classified as a PSEE, SPADE is included to illustrate that a PSEE can be used as a PME provided that it fulfills properties such as being a high level formalism (hiding details), that it is conceptually clear (easy to understand), and provides a diagrammatic representation.

APEL

APEL \([101]\) is currently being developed as part of the PERFECT project. It can be characterized as a generalization and integration between ADELE \([14]\) and Process Weaver \([48]\).

\(^6\)Other PME and PSEE environments: Curtis et al. \([34]\) describe American/Japanese environments including APPL/A, STATEMATE, AP5, GRAPPLE, MARVEL, MVP, Role Interaction Nets, HFSP, SPMS, System dynamics. The first PROMOTER book \([50]\) includes 10 European projects: EPOS \([33]\), SOCCA \([46]\), MERLIN \([74]\), OIKOS \([94]\), ALF \([24]\), ADELE-TEMPO \([14]\), E\(^3\) \([5]\), SPADE \([7]\), PEACE \([1]\), PADM \([23]\). Armenise et al. \([2]\) include selected American, Japanese and European projects: Adele, ALF, APPL/A, STATEMATE, EPOS, FunSoft, HFSP, Marvel, Merlin, MVP-L, Oikos, SPADE.
The APEL v2 environment consists of four integrated editors in which the modeler may describe different aspects of the software process. Each editor offers a different set of concepts and can be used independently from the others. By applying the available concepts we create a model of the corresponding aspect, additionally, each editor provides a possibility to create and browse links between the different models. The set of such models are called project models in APEL terminology. The Product editor allows us to describe our products in terms of product types and their relationships. Concepts include product types, attributes, methods, inheritance, aggregation, and user defined associations. The Activity editor allows us to describe our processes in terms of activities and their relationships. Concepts include activities, input products, output products, workspaces, roles, teams, data-flows, control-flows, and events. The Role editor allows us to describe teams and roles types and their relationships. Roles can be linked by inheritance. A team can be an aggregate of other teams and/or roles. The Workspace editor allows us to describe repository hierarchies, product propagation, and coordination policies.

BM

The Base Model (BM) [23, 107, 108] specification method is developed at the Informatics Process Group at Manchester University, UK. The BM language can be described as a behavioral object-oriented language with objects that provide operations and an encapsulated internal state. The two basic constructs are the single object and the composed object. Objects communicate by calling operations of other components. The behavioral semantics of the BM constructs are defined using temporal logic, and its formal foundation makes it possible to reason about BM specifications mathematically (i.e. deadlock detection, termination and reachability properties). A prototype BM specification tool and a BM “stepper” has been developed, the latter is reported to be useful in finding errors and gaining confidence in model correctness. An extended BM called Organizational Base Model (OBM) is described in [108], here the focus is put on stepwise refinement of BM models.

E³

The E³ [5] PML offers kernel classes and associations which are given graphic symbols. Associations are organized into two inheritance hierarchies, one for binary and one for ternary associations. The kernel associations are Aggregation, Preorder, Feedback, Responsible, Subtask, Input, and Output. The E³ PML kernel classes are Task, Data, Tool, Role, and their superclass E3Object. Following an object-oriented approach the user can create new classes and associations that inherit from the kernel ones.

Associations (not classes) can be defined. A template, that is a general (type level) process model, is the graph whose nodes are classes and whose edges are association definitions. The template origin is the root of the decomposition hierarchy of a template. The origin is the only Task, in a template, which is not a component of another Task, according to an association subtask definition.

A drawing tool enables the modeler to define, browse, query, and reuse process model fragments according to four E³ views which are inheritance, input/output, functional decom-
position (task aggregation), and the information perspective (data). The PML is formally defined: The structural part is defined by means of Z [41] and the dynamic properties by CIRCAL [93]. Dynamic properties of new classes must also be defined in CIRCAL.

IDEFO

IDEFO [91] is a subset of the Structured Analysis and Design Technique (SADT) notation, standardized as IDEFO by the DoD CIM Initiative. An IDEFO model is a set of diagrams where each diagram consists of six or fewer boxes connected by arrows. The boxes represent work activities which can be decomposed into lower level diagrams. There are four “types” of arrows: An arrow entering the left side of a box represents a major input to the activity. An arrow leaving the right side represents a product of that work activity. An arrow coming into the top of a box represents controls or constraints (e.g. schedules) that restrict the way that the activity can be performed. An arrow coming into the bottom of the activity box represents the group(s) that carry out that activity. IDEFO is slightly more powerful than a standard Data Flow Diagram because of its control and resource add-on’s. No particular modeling environment is mentioned in [91].

SPADE

SPADE [7] is a software engineering environment that supports Software Process Analysis, Design, and Enactment (SPADE). The following components have been developed: A PML called SLANG, a PME environment in which SLANG descriptions are stored by means of an object-oriented database, a SLANG interpreter (Process Engine), and a tool integration facility based on DEC FUSE.

SLANG is a high level Petri net based formalism. Tokens and places in the Petri net are typed (Abstract Data Types). SLANG distinguishes between normal places and user places, normal places receive tokens produced by other transitions, user places may receive tokens not produced by transitions. Transitions can either be normal transitions or black transitions representing asynchronous tool invocations. Normal transitions can be activities\(^7\) that represent an abstraction on places and transitions to achieve Petri net modularity.

STATEMATE

SEI\(^8\) have adopted Statecharts [60] and the associated tool called Statemate [59, 60] as a process modeling environment [78, 79]. The SEI software modeling approach uses three modeling perspectives: 1) Functional perspective: Activity charts, enhanced data flow diagrams. 2) Behavioral perspective: Statecharts, shows activation and deactivation of process steps. 3) Organizational perspective: Module charts, shows the organizational units involved in the

---

\(^7\)High-level SLANG adds the concept of activity.

\(^8\)SEI is a U.S. federally funded research and development center (FFRDC) sponsored by the U.S. Department of Defense through the Advanced Research Projects Agency (ARPA). The SEI contract was competitively awarded to Carnegie Mellon University in December 1984.
process, their structure, and the physical communication channels used for information transfer. In addition, the model integrates these three perspectives into a consistent representation of the process, i.e., the structural and behavioral views are subordinate to the functional view. I.e., the functional view includes control activities that correspond to Statecharts, and normal activities can be textually linked to the organizational unit that implements it (a module in a Module chart).

Statemate includes graphical and textual representation, graphical to describe the different viewpoints and textual to describe their relationship, formal definitions, and informal narrative descriptions. Statemate includes the following analysis and simulation capabilities: 1) It ensures that events, conditions, data items, etc., are defined before referenced. 2) Syntax analysis is performed on event, condition, and data expressions. 3) Checking for missing sources or targets of information flows and state transitions, etc. 4) Static checking such as loops in definitions and unreachable states. 5) Dynamic analysis such as reachability, deadlocks, and nondeterminism. 6) Dynamic stepping and animation. 7) Discrete event simulation (utilized in [79]). Finally, Statemate includes reporting and query capabilities.

2.3.2 PSEE systems

This section surveys some existing Process-centered software engineering environments (PSEE’s). The selection criteria are the same as for PME systems.

ADELE

ADELE [14, 15] is developed at LGI, Grenoble, France. Adele can be divided into a kernel, and applications built on top of the kernel. The Adele kernel is based on an entity relationship database complemented by object-oriented concepts, and an activity manager based on triggers. Concepts such as work spaces, user coordination and synchronization policies are defined outside the Adele kernel. The most prominent applications, the Configuration Builder and the Work Space Manager, support software configuration management. In the following we will describe the Adele kernel and the TEMPO process formalism.

The data-model supports object and relation types, organized as two multiple inheritance hierarchies. Types have attributes, triggers, and methods. An aggregate is defined as all objects related to another object by a given relationship. Aggregates must be defined by the modeler, i.e., he may define different sorts of aggregates and associated semantics. Versioning is supported by the branch concept associated to simple objects and their attributes (revision and variant chains).

The activity manager is based on triggers. Methods on objects and relationships can be invoked as part of the current transaction, and will be executed as a sub-transaction. Events arise when a method is executed in the database, methods calling methods will dynamically generate more events. ADELE events are a mixture of events and conditions, a trigger program is executed each time the corresponding event is true. There are four types of triggers: Pre-triggers, post-triggers, after-triggers, and exception-triggers. The set pre-trigger, method, and post-trigger executions constitute a transaction. After-triggers and exception triggers are executed after the transaction is committed or aborted respectively. Both trigger programs
and methods are written in the *Adele language*, a simple imperative language tailored to access database information and navigate through relationships. A novel concept in Adele is *active relationships*, i.e., context-related behavior can be implemented by defining triggers and methods in a relationship that executes on the origin and destination objects of a relationship (note that these are not the triggers and methods defined in the corresponding objects).

In [14], weaknesses experienced with the trigger mechanism are reported, i.e., difficult to understand, trigger execution is not easy to control, triggers are distributed over multiple types for the same action, and there are no high level concepts such as process, work space, connection or planning. To solve these problems, the authors describe an additional layer called TEMPO, aimed at high-level process descriptions and enactment. TEMPO is a PML on top of the Adele kernel, adding the concepts of process type, role, event-condition-action rule, and connection. A *process mode*\(^9\) is defined as a combination of process types. A *process type* has an associated Work Environment, and offers a set of operations on objects. All manipulations made on an object during the process generate events, and rules defined in a process type make it possible to take these events into account in the work environment. In this way one may define protocols for communication and synchronization between various users within a single work environment. To interact between other work environments one needs to define *connections*. A connection is a special type of relationship assumed to be instantiated between pairs of processes. Depending upon the connections, the activity performed inside a Work Environment may or may not interfere with other activities carried out in parallel during the software process. A library of connection relationships are provided, e.g. with semantics such as notify, re-synchronize, merge, duplicate, and share.

**ARCADIA**

Arcadia [117] is developed by UC Irvine and DARPA. Arcadia has a prototype process programming language APPL/A. This extends Ada with “active” (operation-triggered), derivative relations between software objects, triggers propagating updates from one relation to another, optional-enforcible predicates on relations, and composite statements that provide transaction functionality. However, the support for process change (new tools, new policy, new environments) is weak and depends on intricate APPL/A programming. There is no sub-typing. Taylor et al. [117] describe the research rationale and approach taken by Arcadia researchers in investigating environment architecture issues, i.e., process programming research, object management research, and user interface management research.

**ARTICULATOR**

Articulator [92] is a knowledge-based environment for modeling and simulating software engineering processes. Articulator provides a meta-model of software processes, an object-based PML, and an automated simulation mechanism. Strong (or unique) features include an explicit model of instances, querying of process state, and local conflict resolution (through agent-agent interactions). The system architecture consists of five subsystems: The *knowledge*...
base implements the Articulator meta-model by an object-based approach. The instantiation manager manages the relationship between the meta-model, the process models, and their instances. The behavior simulator controls the simulation of a given software process model and creates a process trajectory over a development period. The query mechanism supports logical rules for various types of deductive queries (information found in the meta-model, models, and instances). The knowledge acquisition manager takes a structured description of agents, tasks, and resources as inputs, then translates it into a configured process model. It also assesses the information gathered from software development projects and stores it for later use.

The top level abstraction of the Articulator meta-model consists of three major object types: resources, agents, and tasks. They are linked together through two relations: agents perform tasks and tasks use resources.

- **Resources:** Tasks consume and produce resources, which alter the values of resource attributes. Attributes include: name, current status, functional description, location, ownership, and usage in tasks and by agents.

- **Agents:** An agent represents a collection of behaviors and associated attributes. Individual agents are single entities such as a single developer or a single machine while collective agents have infrastructure defined for a group of agents. Properties include: skills, available resources, affiliated agents, and organizational constraints or incentives.

- **Tasks:** A hierarchy of decomposition may include multilevel nested decomposition, iteration, and multiple selection. At the bottom level of this hierarchy are actions (e.g. LISP functions). Articulator separates between development-oriented tasks (primary tasks) and coordination-oriented tasks (articulation tasks). Properties include: assigned and authorized performers, task hierarchy and execution ordering, schedule, duration, deadline, start time and finish time, and resources planned to be consumed or produced.

Users of Articulator fall into three categories: Software process researchers (our process engineers) define process models, test them by simulation, compare the simulation results to observed development histories, and refine them according to certain criteria. These users may potentially modify the meta-model. Software project managers (our process managers) select or configure an existing software process model which suits their project needs, instantiate it according to their local project situation, and simulate and refine it to create a plan for development. Software developers (our process performers) use Articulator through a CASE environment\(^{10}\). Articulator coordinates activities according to a prescriptive model, it serves as an information exchange center, and it records the development history.

**EPOS**

The EPOS environment [33] can broadly be divided into EPOS-DB and EPOS-PM. EPOS-DB is a client-server database with nested and cooperative transactions against checked-out workspaces containing files. EPOS-DB supports a structurally object-oriented data model to

\(^{10}\) These tools and the enactment mechanisms are not described in the paper. The paper focuses on the architecture, the existing meta-model, and the simulation capabilities.
describe entity and relationship types with single inheritance. The client DML interface is expressed in Prolog. EPOS-PM consists of a PML called SPeLL [31], the Schema Manager [73], the Execution Manager [32], and the Planner [87]. SPeLL is a language for describing EPOS types combining concepts from object-oriented and rule-oriented systems. Types constitute a schema, handled by the Schema Manager. EPOS acts upon expressed goals given by the user. The Planner uses the goal, the product structure, and the task types to instantiate a task network. The task network is a set of objects and links that are instances of types in the schema. The task network is executed and controlled by the Execution Manager. EPOS has focused on evolution as the schema may evolve according to a set of meta-rules, associated to meta-types. Another feature is that the planner may re-plan upon execution failures or type level changes.

MARVEL

The early versions of Marvel [75, 76] focus on the personal software process, i.e., to assist a programmer in his coding tasks by means of artificial intelligence. Particularly, Marvel handles early error checking and answering questions about programs. The user may browse and query object structures, additionally, Marvel undertakes simple development activities so programmers need not be bothered by them, e.g. compile and link.

The key components are an object base containing all entities of interest, and a process model made up by a collection of rules. Each rule represents a single activity, it consists of a name, a list of parameters, a condition that must be true in order to “fire”, the activity to be performed, and a set of mutually exclusive effects in the object base. Rules are implicitly related to each other through matches between a predicate in the condition of one rule and an assertion in the effect of another rule. Marvel applies forward and backward chaining techniques.

A general PSEE architecture that allows for flexibility in selection and application of synchronization mechanisms is described in [16]. The architecture is exemplified by Marvel 3.0. Here we only describe the centralized Rule Processor (RP) and its relationship to client Activity Executors (AE). The rule processor is placed on a server while the activities are executed on the client. Process enactment in the rule processor is done as follows: When an activity is requested, the condition of the corresponding rule is evaluated. If not satisfied, Marvel attempts to satisfy it by backward chaining to other rules whose effects may satisfy the user invoked rule. This is done recursively until the condition is satisfied or all possibilities are exhausted. If satisfied, the corresponding activity is performed on the client. When the activity returns from the client, the rule processor asserts the effect indicated by the returned status code. The rule processor then recursively forwards chains to all rules whose conditions have become satisfied.

IPSE 2.5

IPSE 2.5 [102] can be characterized as an executable PML with roles, interactions, and actions as the main modeling concepts. It is also known for its reflective properties, i.e. a process model can be inspected and changed at run time. The PML is an object oriented conceptual
modeling language. A Class is defined in terms of a set of properties organized into a set of property categories, i.e. resource, associated roles, actions, start and stop conditions. The prototype environment (Simulator) was implemented in Smalltalk, it allows creation, modification, and execution of models written in the language.

IPSE 2.5 has been further developed and commercialised as Process Wise Integrator [23]. BM is a specification method for IPSE 2.5.

PROCESS WEAVER

PROCESS WEAVER [20, 49, 48] provides two tools for process modeling, the method editor and the cooperative procedure editor. The method editor provides functionality to define a hierarchical set of activities, each activity with a list of inputs, roles, and outputs. Activities may have an associated cooperative procedure that defines the activation sequence of lower-level activities. The cooperative procedure editor provides functionality for defining processes, using a Petri-net metaphor. Basic control structures are modeled as places and transitions, a transition being divided into condition and action. There is a predefined set of conditions and actions that can be regarded as templates that are refined by means of user defined logic and actions in the CoShell programming language. Atomic work is modeled in a separate tool named Work Context editor.

The Agenda is a window on the user’s screen. In this window an icon will show up when the user is assigned a certain task. When the user clicks on this icon, a work context will show up. This is a window containing tools and information the user needs to do his assigned work, e.g. documents, word processors, spread sheets. When the user finishes his work, he clicks his done button to report to Process Weaver that he is finished.

The Activity Instance Manager (AIM) is a tool that supports project management activities. The AIM work on a project file, this file is established when running the predefined cooperative procedure. Among other things, the user selects a PW Method to be used in the project. The Method is copied into the AIM and initially presented as in the method editor. However, in the AIM, the user will instantiate activities into unique tasks. With the guided option this means to invoke a cooperative procedure, with the manual option it means to change the status of a task. Procedure inputs and outputs can be propagated between parents and child upon procedure invocation and termination. The graphical interface shows tasks and their status. The AIM offers functionality for tracking effort used on a task, to trace events occurring in a task, and to make reports about tasks in a given state. The AIM method can be exported to a file and then imported into Microsoft/Project (MSP). A task network can be created and scheduled by means of MSP and then brought back to the AIM for execution. Also, effort reported by means of the AIM can be exported to MSP for aggregation and presentation.

SPADE

The SPADE environment [7] is also described above, here we look at process enactment features. The SLANG description is stored in an object-oriented database, a process fragment (activity) is executed by making a copy (active copy) that can be executed by the SLANG
interpreter. The active copy has an associated state (marking), the interpreter monitors
the guards and fire transitions, consuming and producing tokens. User places are used to
capture tool events, i.e. events caused by user actions. An interesting feature of the SPADE-1
prototype is the strong separation between user interactions and process model interpretation.
The user interaction environment is responsible for performing the interaction between users
through tools in the environment, while the process enactment environment is responsible for
executing the process model. SPADE implements a filter that manages the communication
between these environments, the filter adds functionality to the FUSE MSG Server.

SPMS

The Software Process Management System [84] (SPMS) is developed within the context of
a DARPA/STARS program. Interesting features of SPMS includes: Process modeling inte-
grated with process management, process modeling integrated with product quality metrics,
synthesis of reusable process components, and simulation capabilities. Krasner et al. [84]
state that *SPMS enables process-driven development with automated assistance based on liv-
ing models of the process*, however, process enactment capabilities are not described in [84].

A SPMS process model is represented as an activity-on-node network. SPMS separates
between process primitives and process components. Process components are named sub-
networks within the larger model, process primitives are the individual nodes and links
within a process component. Supported process primitives are: Tasks, milestones, prod-
ucts, AND/OR precedence logic, validations, temporal constraint links, and phase summary
links.

SPMS distinguishes between the generic model (type level), and the actual plan (instance
level), both with graphical support. The relationship between type and instance models
is not properly described, but it seems to resemble the approach taken in EPOS. E.g., it is
indicated that the instance network is built on the basis of a product decomposition hierarchy,
and that a hierarchical and incremental planner is applied. Also similarly to EPOS, SPMS can
be classified as a hybrid approach, applying both object-oriented and rule based techniques.

2.4 Experiences

2.4.1 Process modeling

Several papers have been published on successful modeling of organizational processes (quality
manuals, handbooks, methods). Since PSEE’s have been used as PME’s we include such
experiences as well.

ARTICULATOR

Votta [118] describes an experiment at AT&T Bell Laboratories. The goal was to compare
one formal and one informal process description. The informal development process (AT&T
On Line Methodology, OLM) contained 71 process descriptions, 51 document outlines and
templates, 34 procedure documents, and other textual developers aids. Two processes, Feature Design and Document Review, were selected for formalization. The experimental design was constructed with two teams: the conversion team was responsible for generating the Articulator models from the OLM, while the evaluation team, consisting of experts from the development organization, judged the result. The conversion team constructed the four different Articulator views – task, agent, product and resources – in 6 weeks.

The evaluation team compared the two representations and found the Articulator approach to have the following advantages [118]:

- The representation is more compact and complete than the one that currently exists in the OLM using English prose – that is, it is more succinct.

- The economy of representation has the side effect of improving the capture of process state information – that is, it is more accurate.

- The formalism requires critical information that is currently missing in the OLM – that is, it is more complete.

- As a result of the succinctness, accuracy, and completeness, the representation is easier to understand and easier to integrate with other processes.

- Finally, the accuracy and completeness enables a more precise identification of significant events, and hence, the determination of what should be measured – that is, it is more measurable.

Furthermore while building models, the conversion team detected deadlocks and unreachable process states in the OLM description. Votta points out that such problems is a significant source of process exceptions. Finally, the evaluation team stated that process management would be supported by a more easy identification of significant events, and project management by removing irrelevant milestones.

The Votta paper addresses an important question: Why is the development organization not converting to a formal process description – yet? The evaluation team suggested three major areas of concern: 1) Scaling: Does the tool scale up? 2) Return of investments: Will conversion cost be recovered? 3) Organizational aspects: Can the organization be re-engineered so the benefit of using these tools is realized? These are open questions in which no further work has been reported.

BM

Sa and Warboys [107] report experiences from using BM to describe a software design process. It is reported that BM was found to be useful in providing an abstract but yet precise description of a process. In [23], experience is reported on the specification and validation of the ISPW-6 reference example in BM. Furthermore, the ISPW-6 implementation in the Process Wise enactment language is described. The paper also describes how the consistency between the BM specification and the Process Wise implementation can be checked. Both the validation of the BM specification and the verification between the BM specification and the
Process Wise implementation was done by means of executing scenarios with a BM stepping tool.

$E^3$

Baldi and Jaccheri [4] report experiences from using $E^3$ [5] to model the process manual of FIAT/Iveco. The input of their work were 175 pages of text and pictures, and the output were 161 $E^3$ classes and 585 $E^3$ associations. The main lesson learned was that errors and inconsistencies were found due to the application of the $E^3$ environment (method, language, and tools). More specifically, they found that $E^3$ contributes by forcing task sequencing and task input/output to be explicit, that formality contributes to finding inconsistencies, and that $E^3$’s storage, view, and query mechanisms facilitate human understanding and validation. Problems revealed in the Iveco process manual were classified as inconsistency, ambiguity, redundancy, and terminology problems. Problems revealed in the $E^3$ system were the lack of a redefinition mechanism for associations upon inheritance of class clusters, the lack of a flexible view mechanism, and the lack of real simulation capabilities.

IDEF0

McGowan and Bohner [91] describe Model Based Process Assessments (MBPA), an approach to combine process modeling with process assessment. The paper reports experiences with an MBPA project utilizing IDEF0 as a PML. The focus is put on a MBPA project assessing the maintenance process for Contel’s Carrier Access Billing Service (CABS) system (1.4 millions lines of code, maintenance staff of 40 information system professionals). The success criteria were for CABS to stabilize its request backlog and to move from reactive to proactive maintenance. The paper describes how the SADT diagrams were used in the process elicitation process, i.e. interviews and reviews. The entire CABS model consists of 12 simple diagrams, each with 6 or fewer activity boxes. The MBPA project delivered a written report that contained (a) the SADT model of the maintenance process, (b) observed strong features, (c) recommendations for change, (d) a model of the improved process. It is reported that the CABS organization acted favorably on all of the recommendations and that within 6 months there were some impressive results on increased process speed, decreased backlog, and decreased staff size. The main experiences were that SADT was an effective medium for communication about the process, that the provided modeling concepts were sufficient, and that a process model leads to effective process improvement recommendations that other improvement approaches often miss (e.g. process boundaries, overlapping activities).

PROCESS WEAVER

Aumaitre et al. [3] report experiences from modeling the complete software development process at the European Space Agency. 170 pages of text, tables, and data flow diagrams were formalized and implemented in Process Weaver [48]. The validation was planned to be

---

11 MBPA is seen as an alternative to the SEI software process assessment (SPA).
performed by enacting it in a simulated environment, however, the results were not available when the paper was written. The lessons learned are:

- Tools are needed for constructing and navigating within a large process specification to help ensure its consistency and completeness.
- Systematic requirement capture methods are needed to support the construction of a process specification from an informal or implicit definition of the process.
- Redundancy in the specification creates a consistency maintenance problem.
- Different users need different representation formalisms.
- There is a need for parameterized process definitions.

The implementation in Process Weaver required an effort of six person months and created 300 activities and 400 work-contexts.

SPADE

Bandinelli et al. [8] report experiences from the use of SPADE’s [7] PML (SLANG) to formalize an existing, informal process description. The process modeling was done at an Italian telecommunication company (Italtel, Business Unit Telecommunications for Defense (BUTD)). 200 pages of BUTD’s Quality Manual were formalized into 24 SLANG activities, including the total of 210 places and 180 transitions. The study lasted for about six months and the interaction with the process owner was limited to answering about 60 questions and ca. 10 hours of meetings. The hypothesis is formulated as: “The basic assumption that has generated and driven the work discussed in this paper is that the adoption of formal process modeling languages can help reduce the divergence among the Desired, Official, Perceived, Observed, and Actual process.”. Furthermore, the authors state that “To prove this hypothesis, CEFRIEL and BUTD decided to use process formalisms to model a significant part of the BUTD software process,...”.

The main contributions are stated to be: 1) The introduction of different views on the software process (desired, official, perceived, observed, and actual). The findings of the study are described in the light of these perspectives. 2) A solution to the problem of interacting with people that cannot be distracted from their work. 3) To show that the adoption of formal PML’s can help reducing errors and deficiencies in the documentation. The latter is exemplified by: a) Inconsistencies: The same concept described by conflicting statements in different parts of the Quality Manual. b) Incompleteness: Entire parts of the process description are missing. c) Ambiguities: Leave choices between different alternatives open. d) Poor organization: The process documentation needs restructing to improve readability and clarity.

The belief or disbelief in the hypothesis is not commented on in the paper, although one can read between the lines that the authors think that the hypothesis is supported.
SPMS

Krasner et al. [84] report experiences from the use of the Software Process Management System (SPMS). The first experience is from the formalization of the STARS Software First Life Cycle (SFLC) definition (80 pages, developed by IBM), as well as the IEEE Working Group Draft Standard P-1074 process fragments. Trial use of the SFLC model on the SPMS project included 32 defined software components and resulted in a project-specific model with 3,500 process primitives (nodes and links). Lessons learned: 1) The importance of a graphical means of entering, browsing and editing the process model components. 2) Automatically generating project specific process models can produce large volumes of data that may require further processing. 3) The SFLC process was not sufficiently detailed to support automatic tool invocation, but was at the level necessary to support task planning and scheduling. 4) Process model representation should be compatible with tools for software cost and scheduling. 5) The SFLC process varied in level of detail with respect to the different processes, and different levels of abstraction were often found to be inconsistent or ambiguous. 6) A role view of the process model was found to be helpful in initially obtaining and validating process models. 7) Measurement and analysis models must evolve with the software development process models if these models are expected to be integrated into a self-monitoring enactable process management system.

The SPMS team also modeled a Cleanroom process description manual containing 125 pages of Box Structured notation and textual descriptions (SEI Process Asset Library project). Due to the high-quality of the process manual, the modeling was done in 20 person days, resulting in 25 subprocess groups, 106 activities, 200 decision points, 70 stimuli/responses, and 16 state data artifacts. Lessons learned: Generally, SPMS was found to handle behavioral, functional, and organizational process information well. Functional enhancements: 1) Support for additional process semantics, i.e. process naming convention and typing. 2) More general interoperability of data. 3) Better visualization support. 4) On-line help. 5) Automatic component inheritance of attribute values. 6) User-definable metrics. 7) Object state data tracking. 8) Better support for cut-and paste subprocess modeling. 9) A more complete precedence logic, i.e. NOT operator needed.

STATEMATE

Kellner and Hansen [80] report experiences from using STATEMATE for the purpose of process modeling at U.S. Air Force. The investigated process was the processing of Technical Orders with focus on updating the associated documentation. The first approach was to produce a narrative textual description, and a high level DFD to increase readability. The authors describe the shortcomings of the DFD approach, i.e. that it only shows a functional viewpoint, it has limited expressive power, it does not show how the process is implemented in the organization, and lacks processing power to be automatically analyzed.

The authors then formulate a set of requirements for an ideal approach to software process modeling which resembles the ones published in [78]. Based on these requirements the authors evaluate some project management toolkits, expert system packages, simulation toolkits, MIS toolkits for analysis and design, toolkits for analysis and design of real-time reactive systems.
The authors decided to use STATEMATE, an example of the last category, as they provided a mapping of provided concepts to process modeling needs. The results from the modeling is reported to be increased understanding of the process by participants and managers and a series of recommendations for changes (improvements). Lessons learned was that “real-world” processes could be surprisingly complex, that several process participants should be involved in the process elicitation (i.e. interviewed), that the modeling activity was complicated by the fact that the process had not been executed repeatedly, and the importance of gaining understanding of the purpose, goals, and objectives of the overall process.

2.4.2 Process enactment

The only reported real-world experience is the Process Weaver paper [20]. The Marvel paper [9] is included because it use data from real-world processes, “simulating” the real-world environment.

Process Weaver

Bronisz et al. [20] describe the introduction and use of Process Weaver internally at Cap Gemini Innovation, as part of a dissemination program. The goal of the dissemination program is stated to be to provide measurable benefits to the company from improved processes, to gain experience in the introduction of Process Weaver in the daily work, and to evaluate its usefulness. The paper reports that a set of administrative processes were implemented as cooperative procedures. Specifically, the paper reports experiences with the Monthly Report process. This process collects the monthly progress reports from all projects at the different branches of the company, the reports are compiled and assembled into a global company report. The process involved several people, its main problem was that it was often incomplete and delayed. Before the introduced process support the delivery date had been the 10th of the following month, and included 60% of existing projects. After the introduced process support the delivery date was the 2nd, and included 90% of existing projects. Although not backed up by empirical data, it is claimed that people also saved time performing the process, and that the Monthly Reports were more frequently read. The paper does not report any negative findings with respect to Process Weaver or its application.

MARVEL

Barghouti et al. [9] report experiences from two case studies at AT&T applying Marvel 3.1 as a process modeling environment. The first process, the Signaling Fault Resolution (SFR) process is reported to be a computer intensive, routine operation maintenance process. The second process, the Initial Modification Request Resolution (IMR) process, is reported to be a human-intensive software maintenance process. Both processes were described by means of Marvel’s process modeling language (MSL), in the first case from scratch, in the second case by formalizing an existing informal description. After developing the models in MSL, Marvel was used to generate process enactment environments for the models, and then populated with historical data. With the sample data the authors ran scenarios “simulating” the actual
execution of the processes and verified that the enactment of the process model performed as expected.

An interesting reported experience is that in the IMR case the authors had to disable Marvel’s forward and backward chaining mechanism because tasks required human intervention and could not be started automatically, e.g., that a coding task could not be automatically started upon programmer assignment. Their conclusion is that “Human-intensive processes are not as amenable to automated support as computer-intensive processes”. It is further reported that the modeling activity revealed three kinds of problems in the informal specification: 1) Missing inputs and outputs of tasks. 2) Ambiguous input and output criteria for tasks. 3) Inefficiency in the process definition.

The major limitations of Marvel 3.1. was found to be: 1) “Coding a model of a realistic process in MSL is cumbersome and requires more modeling sophistication that is present in many organizations. A higher-level graphical notation may be more suitable, especially during the early phases of process modeling.” 2) “Closed systems, such as Marvel, force users to change the way they interact with their tools.” 3) “Marvel runs only under the UNIX operating system.” The latter limitation is pragmatic and a matter of development resources, the former two are more fundamental and interesting as they say something about important PSEE problems.

The authors state that “we showed that the same process modeling language and process enactment engine can be used to model and enact computer intensive processes like the SFR process as well as human-intensive processes like the IMR process.” Seen in connection with the limitations mentioned above, this must be interpreted as “the researchers capability to express their understanding of the process by means of the MSL language”, as there are no indication in the paper that the models were validated or used by actual process participants.

2.5 Preliminary answers to overall questions

All the overall questions are asked within the context of software development processes performed within the boundaries of a project. Furthermore, it is assumed that a configuration management system is in use, and that every artifact produced by the project is stored as an SCM item.

Process modeling – industrial process

Q1: Can existing modeling support (PML and PME) be used for industrial processes? (technical feasibility)

To the extent that process handbooks represent industrial processes, the answer is yes. Existing PML’s which have been used for the modeling of industrial process handbooks are: Articulator[118], E3 [4], MARVEL [9], Process Weaver [3], SPADE [8], SPMS [84], STATEMATE [80].

Few have been concerned about whether these handbooks correctly represent the industrial process, and few have put much effort in model validation. One paper, namely IDEF0 – CABS
[91], is concerned about process elicitation and validation in terms of iterations between interviews, modeling, and model presentation. This work has not been concerned about collecting evidence from executed projects. Thus, there is a danger that the developed models represent an idealized or wanted process, one that has never taken place.

Q2: What are the benefits of using existing PML’s and PME’s?
(technical/economical feasibility)

Most reported experience are variations on the same theme, namely consistency and completeness of the process handbook descriptions. The most thorough list is reported by Votta [118] which states that the ARTICULATOR model has the following advantages over the existing OLM description: Succinctness, accuracy, completeness, easier to understand, easier to integrate with other processes, more measurable, and deadlocks and unreachable process states could be detected. The conclusion is that the quality of the process manual is improved by using a formal PML.

Q3: Is software process modeling cost effective?
(economical feasibility)

No published paper has given an answer to this question. Votta expresses uncertainty in [118]: Will conversion cost be recovered? The paper does not state the cost of the conversion activity, neither the relative size of the formalized part of the OLM method.

Process enactment – industrial process

Q4: Can existing enactment support (PPL and PE) be used for industrial processes?
(technical feasibility)

Bronisz et al. [20] report that Process Weaver has been used for administrative processes within CAP, but not in software development processes. The paper reports experiences with the Monthly Report process, which is not a software development, and does not address project processes or the use of configuration management systems.

Barghouti et al. [9] address software development processes but not in a project context, and not the integration with configuration management systems. The conclusion of the Marvel experiences reported in [9] are: 1) They had to disable Marvel’s forward and backward chaining mechanism because tasks required human intervention and could not be started automatically. 2) “Coding a model of a realistic process in MSL is cumbersome and requires more modeling sophistication that is present in many organizations”.

Q5: What are the benefits of using existing PPL’s and PE’s?
(technical/economical feasibility)

Bronisz et al. [20] report that an administrative “monthly report” process implemented in Process Weaver caused quicker and more complete reports, saved time in performing the process, and the report was more frequently read.

A synthesis of assumed benefits reported in the process modeling literature can be summarized as follows:
• **Process guidance.** Process guidance can be divided into active and passive guidance. Active guidance happens when the process environment proposes a next action. Passive guidance is when the user must take action by asking the process environment in one way or the other. However, to qualify for guidance the user must be capable of ignoring the advice, else it is enforcement.

• **Process enforcement.** A process is enforced if the corresponding work can only be achieved by accepting the possibilities offered by the process program. The process program encompasses decisions about the level of detail (detailed, broad, strict, free, etc.) but there are certain boundaries that cannot be exceeded.

• **Process automation.** Parts of the process is hidden from the process performer, e.g. tool invocations, collecting data from a repository. Tedious and repetitive work can be avoided because process knowledge can be embedded in a process program.

• **Process coordination.** Facilitate data and event propagation between users, e.g. we want to receive a notification when a product is checked-in to the repository.

**Q6:** *Is process enactment cost effective? (economical feasibility)*

There are no reported experiences with respect to software processes. The Bronisz paper [20] does not describe implementation costs of the Monthly Report process. Neither does there exist any follow-up paper on the use of the Monthly Report process. Such a paper would have been interesting because it could give an indication of both usability and cost effectiveness, as the belief in usability is strengthened by continuous use, and as the pay-back increases with the number of process executions.

**Q7:** *Can existing instantiation support be used for industrial processes? (technical feasibility)*

For single, non interacting processes the answer is yes, e.g., as exemplified by the Monthly Report process. With respect to software development processes within a project context, the question is unanswered. Additional problems in a project context is the problem of fragment composition and interaction. Fragment composition can be exemplified by project planning, and fragment interaction by the need to bind input and output products between different process fragments (e.g. through the configuration management system).

Project planning is discussed by Krasner et al. [84], but they do not report experiences on existing or historical projects. Krasner reports SPMS models which are instantiated into example project plans, furthermore, it is unclear whether SPMS supports enactment or not.

The conclusion of the Marvel experiences reported in [9] are that they had to disable Marvel’s forward and backward chaining mechanism because tasks required human intervention and could not be started automatically. Since Marvel relies on forward and backward chaining mechanisms for instantiation, the instantiation support in Marvel is reported not to be adequate for industrial processes.
Process modeling – process enactment

SCM and PSEE systems are only concerned about products and processes that can be handled by computer programs. Process modeling in general is concerned about all aspects of the software development process.

Q8: Can every modeled process be enacted?
What are the modeling/programming boundaries?

Automation boundaries have not been discussed in the process technology literature. The only identified experience is reported in [9], stating that “Human-intensive processes are not as amenable to automated support as computer-intensive processes”. This can be interpreted as Marvel being suited for software build processes (e.g. compile and link), but not general software engineering processes which take place in a project context.

Q9: Can every programmed process be enacted?
What are the programming/executation boundaries?

The simulation of processes as reported in [9] does not mean that processes can be executed in a software project. E.g., there may be assumptions in the simulation/stepping that do not hold in a real-world context. The disabling of Marvel’s forward and backward chaining mechanism is an example of something that was programmed but turned out not to be executable. In the large, the question is unanswered.

Q10: Should the same language and environment be used for both modeling and enactment?

Examples of languages and environments that originally were conceived for enactment but has been used for modeling are Articulator [118], MARVEL [9], Process Weaver [3], and SPADE [8]. The reported experience is that higher level formalisms are needed, but all state that this can and should be provided within the existing system, i.e., be translated to lower level execution mechanisms.

BM and E³ are examples of pure modeling languages and environments. The distinction between a PML for the purpose of modeling and a PML for the purpose of enactment is discussed in [107], and the overall remark is that this distinction is not clear cut. In [107] it is stated that such a distinction seems to be useful, e.g. that a PML for the purpose of modeling represents a “user view” on the process, that the focus is on high-level issues, and that properties of the model can more easily be deduced.

Process modeling – SCM

Q11: Which are the relationships, if any, between process modeling and SCM?

The question is not answered by the existing literature. The explanation can be that it has no interest or relevance, or that process researchers are not aware that the existence of SCM system might or should influence process modeling activities and models.
Process enactment – SCM

Q12: Can PSEEs replace SCM systems?

This is not discussed in the literature. Most PSEEs do not provide SCM functionality, e.g. SPADE, Process Weaver, Articulator, Marvel, and Process Wise. Two PSEE’s provide integrated SCM support, namely ADELE and EPOS. No industrial experiences are reported on PSEE/SCM integration.

Q13: Can PSEEs be used together with any SCM system?

This is not discussed in the literature. Papers on PSEE systems such as SPADE, Process Weaver, Articulator, Marvel do not mention this even as an area of concern, and one can be tempted to conclude that they can be used on top of any SCM system.

Q14: If not, what are the requirements for an integrated system?

If answers to both Q12 and Q13 are negative one can conclude that a PSEE without SCM functionality cannot be applied to software engineering projects. Furthermore, that the only viable strategy for a PSEE intended for software projects is to be integrated with a SCM system. Therefore, integration requirements are of interest.
Chapter 3

The studied case

Our case is the process descriptions developed and used by an organizational unit, the unit’s usage of a configuration management system, and the relationship between the process descriptions and the configuration management system. Our goal is to gain experiences in software process modeling. Our method to gain experiences is to model the case by means of a process modeling language. This chapter describes the studied domain and defines the studied case. Chapter 4 describes the applied modeling language and the developed models. Chapter 5 describes modeling experiences.

This chapter is organized as follows: Section 3.1 gives an overview of the case study and the studied case. In Section 3.2 we characterize the organizational unit in which the study was performed. In Section 3.3 we describe the existing process descriptions of the organizational unit. In Section 3.4 we describe the CM system used by the organization. In Section 3.5 we describe how the organization applied the configuration management system.

3.1 Case study overview

The case study was performed within a software development department of a larger company, lasting from August 1994 until December 1995. The case study context is described in section 1.7, page 9. Within this context our case can be defined as follows:

- A particular project within the Company, denoted as PRO1.
- A particular version of the Engineering Handbook developed and applied within the context of PRO1. This handbook contains the process descriptions of PRO1 development processes.
- Product structures and definitions in the applied configuration management system regarding the particular product developed by PRO1.

The Engineering Handbook and reports on CM product structures were made available to this study by the Company. This information was used as input to a process modeling activity. The process modeling activity was performed by the thesis author. Additionally, some work
was done by a Diploma student supervised by the thesis author. The role of the Diploma student was to install and maintain a prototype of the APEL environment, to document the developed models in this tool, and to evaluate the tool functionality. The tool evaluation can be found in [57] and in [62].

During the process modeling activity there were several meetings with Company personnel, and furthermore, there was communication by means of fax and e-mail. The Company used several hundred hours on supporting the modeling activity throughout 1995, facilitating the work to create models that are detailed and close to actual development practices.

Presenting process models is a balance between not showing company confidential information and wanting to show interesting examples or explanations. This is problematic in a case where models are detailed, correct, and used. We have taken the following actions not to show company confidential information:

- The \textit{product} is protected. The developed product is never described in terms of its functionality or purpose. Neither are any real product names mentioned.

- The \textit{personnel} is protected. The developers are never mentioned by names or by abbreviations.

- The \textit{process} is protected. Processes are only partially described. The criterion used is to describe enough to understand the corresponding diagram, but not to be able to perform the process. Furthermore, product type abbreviations are never explained, and only a few of the product abbreviations are expanded into full names.

The process models in the next chapter are presented as drawings and not as diagrams from the modeling tool. This is because of tool problems with respect to generating postscript code.

\section{Characterizing the organization}

The studied case is the process descriptions and the configuration management practices of a particular software development group in the Company. The group consists of approximately 15 software professionals and can be characterized as producing software for telecommunication purposes, usually delivered as embedded software. At any time there are about three concurrent software projects, with a typical size of about 7000 person-hours. Projects usually follow a slightly modified waterfall life-cycle model, that allow certain activities to be run in parallel. E.g., software projects are often run in parallel with hardware development and the two activities are merged during the system integration activity. Most of the projects apply the SDL\footnote{Functional Specification and Description Language, standardized by CCITT} specification language and a case tool called SDT.

The software development group applies a rigorous configuration management regime that includes all artifacts available in an electronic format. Hereafter, we will denote these artifacts as \textit{documents}, even though they also include source code, test results, project plans, etc.
The configuration management system used is PCMS [115]. PCMS includes most of the functionality expected from such a system, e.g. revisions and variants, baselines, system builds, product structuring, etc. PCMS had been in use for some years in the Company before our work begun.

The software development processes are documented in the Company’s Engineering Handbook, including the Change management and Document review processes. Among other things, these descriptions included rules and guidelines on how to apply PCMS in these processes.

The process samples used in our experiment were collected from a particular project within the Company. The total effort used on this project was 7055 hours. The project performed a total of 102 CM processes (1422 person-hours) and 71 DR processes (640 person-hours). Altogether, the two processes represent a significant amount of work (2062 person-hours, or 34% of the total effort).

The change and review reports of this project were made available to the experiment designers. Altogether, the CM and DR process descriptions, as well as the PCMS product structure were made available to the process implementors. This information is sufficient to reconstruct the processes like they were performed within the Company.

### 3.3 The Engineering Handbook

This section describes the Engineering Handbook, its use, and the interplay between the Generic Engineering Handbook and project specific Engineering Handbooks.

#### 3.3.1 The Generic Engineering Handbook

The role of the Generic Engineering Handbook (GEH) is to act as a summary of experiences and as a template for the project specific Engineering Handbooks (EHs). When a new project is started a new EH is instantiated from the GEH taking project specific characteristics into account. Ideally this is to be done by the project manager in cooperation with quality assurance personnel. Moreover, the project manager is to be responsible for maintaining the EH, and possibly to give feedback to the GEH during project execution. When the project is finished the GEH is to be updated by QA personnel, documenting experiences from that particular project.

#### 3.3.2 Engineering Handbook overview

The EH is implemented as a set of documents, altogether about 200 pages of text and drawings. It is organized into an introduction document, a process overview document, a document for each of the nine development phases, and a set of guideline and standard documents.

---

2Product Configuration Management System is a trademark of SQL Software Ltd, UK
• **Introduction.** Contains a description of the purpose, organization, and use of the Engineering Handbook.

• **Overview.** Contains an overview of the content of each development phase and how they are tied together. It also lists abbreviations used in the handbook, mostly concerning document types.

• **Phases.** Contains a description of how to perform the work in a development phase. The phase is decomposed into activities that might be further decomposed, while leaf-level activities have an associated textual description. Each level of decomposition is described in terms of partially ordered activities, the ordering being mainly described in the form of data dependencies (product types).

• **Guidelines.** Contains a description of procedures or techniques that are applicable to several activities, often independent of any particular phase. E.g. how to develop message sequence charts.

• **Standards.** Contains a description of syntactical and representational standards associated to a language or a tool. E.g. rules for how to use SDL and SDLT.

The EH is mainly about defining development activities, a description of what developers should do. Sequencing of activities is usually described in a broad sense, sometimes quite detailed as in the Change and Review processes.

The EH is mainly advocated by the Company **method group**, filling the roles of both the SQA and the SEPG groups as described by Humphrey [64]. Currently the method group consist of a full time **method responsible** and one additional 50% assigned software professional.

### 3.3.3 Process descriptions

The Engineering Handbook describes the software process by process diagrams and textual descriptions. The text describes the work to be performed within each activity, input documents used by the activity, and output documents produced by the activity. Each document type is identified by an abbreviation, and described by a full name and a document template.

![Process diagram](image)

**Figure 3.1: Process diagram**

The applied diagrams are similar to data-flow diagrams (DFD). The style of diagrams is depicted in Figure 3.1. The diagrammatic language is not formally defined, but the Company associate more or less the following meaning to these diagrams:
• **Input:** A list of document types. The activity may start when one or more documents of the given type(s) are available.

• **Output:** A list of document types. An activity finishes when one or more documents of the given type(s) have been produced.

• **Concurrency:** Activity_1 executes in parallel with Activity_2.

• **Sequence:** Activity_3 executes when both Activity_1 and Activity_2 have finished.

We observe that data are used as control, when output documents are available then succeeding activity may start. Two processes, the *Document review* and *Change management* processes are modeled in more detail:

• State transition diagrams for documents depicting PCMS life-cycles, i.e. changing values in a system defined attribute.

• Iteration and alternatives are depicted in the diagrams.

The process descriptions of these two processes were strongly influenced by functionality in the configuration management system, i.e. by document life-cycles. The main problem encountered in the process descriptions were lack of precise chaining and instantiation semantics.

### 3.3.4 Instantiation

Instantiation is the process of making a project specific EH based on the generic one, the project specific EH becomes more or less a subset of the GEH. The instantiation process is basically to make a copy of the appropriate documents and to remove unnecessary parts. Additionally, activities that are known to be unique to a particular project are modified or added.

A single GEH is considered to be sufficient as long as the projects are of about the same type, currently "SDL" projects, that is projects where SDL is used as the main development tool. The Company operate with a before and after SDL schism, and state that projects run before the introduction of SDL are not comparable to the current project profile.

In the EH introduction it is stated that the project manager is to be responsible for EH instantiation while being supported by the method group. It is recommended to do the main part of the instantiation job in the project invocation phase, but to complement with phase details upon phase start-up. However, this must be seen more as a goal rather than the current practice, as of today, the method group more or less do this work.

### 3.3.5 Maintenance

The goal of the Company is to let the EH be maintained by the project manager during project execution. It is an open question if this can be achieved, as instantiation and maintenance is now done by a separate 50% assigned resource. The instantiation and maintenance of the EH
has (in some cases) turned out to be difficult when this person was loaded with other project specific work.

A separate, unofficial Project Handbook was de-facto implemented by a Company project manager. This need was unexpected by Company process personnel who thought that the EH covered this need. This project handbook included information about named products, tasks, and resources, i.e. what we call “instance level information”.

3.3.6 Use

It is not mandatory, but recommended for project participants to read the EH. The way the EH is written, its main purpose is to give practical hints and guidelines for project participants, not to be a tool for process enforcement. However, various parts of the EH have very different maturity. Some of the phases (detailed design and integration) and processes (change and review) are described in much more detail than others, and are also found to be more complete and consistent. The main characteristics of these processes are that they have been executed more often, and are better understood. We can say that the quality and detail of the process description reflects the level of maturity.

Another interesting point is that broad level descriptions are of more interest to project external personnel than to project participants. Project participants use the EH to the extent that it provides detailed knowledge of how things are done (guidance) or procedures that must be followed (enforcement). An example where the description is often checked is when a project participant need information about “when to change status” of documents in the Change management and Document review processes.

The project manager is required to be familiar with the EH, the intended use is broad and detailed project planning. However, there are many “should’s” when it comes to the relationship between the project manager and the engineering handbook. The method responsible advocates the EH, but it does not seem to have been fully accepted by project managers.

3.3.7 Feedback

In the GEH introduction it is stated that the method responsible is responsible for GEH maintenance. The goal of this activity is to ensure that the GEH reflects experience accumulated in the different projects.

Feedback is the process of updating the GEH based on project experiences. The project manager is to evaluate changes done to the EH and notify the method responsible if they seem appropriate for GEH inclusion. The main feedback comes when a project is finished, but can also be caused by the startup of a new project. In the latter case it might be useful to update the GEH according to EH’s of ongoing projects before it is used for a new instantiation.
3.3.8 Problems and goals

We studied two EH sets that represented the status of the EH with one year of evolution in between them. In this period we found that there had been drastical improvements both in completeness and internal consistency. However, there are still some quality problems:

1. Not every phase was described in a separate document.

2. Inconsistencies between drawings and textual descriptions. E.g., text stating that the EH is produced incrementally before each development phase versus a diagram showing that it is produced in the project invocation phase.

3. Inconsistencies between drawings, e.g., between the input of a parent activity and the input of its children.

4. A single product type was denoted by different names in different parts of the document.

These kinds of problems are roughly the same as those reported by other researchers such as [3, 4, 8].

In a period of 2 years the Company have used approximately 1100 hours on GEH and EH creation and maintenance. A man-year is about 1800 hours, and this gives us an effort of 5/8 man-year on process modeling activities. If we take a specific project (PRO1) we find that the effort used on the EH corresponds to about 8% of total project effort (547/7055). Compared to Humphrey’s 2% guideline on SEPG effort the 8% seems far too high to be accepted. It is also judged to be too high by Company personnel, but they have not decided on what the figure should be. However, it is reasonable to expect the figure should not exceed 1%, considering that other SEPG activities such as metrics are also a part of Humphrey’s 2% figure. If this holds, the Company process modeling effort should be reduced by a factor of seven while maintaining or increasing the quality.

The main goal of the Company with respect to the EH is to reduce the effort needed for instantiation, maintenance, and feedback, while maintaining or increasing the EH quality. This is the Company’s motivation for looking into formal process modeling languages and supporting environments.

3.4 The configuration management system

This section describes the configuration management system used by the Company. We have chosen to describe the main concepts in this system because the Company’s processes are strongly influenced by its use. Furthermore, in the enactment experiment described in Chapter 7 we are applying this CM system. A basic understanding of this system is also necessary to understand experiences reported in this work.

This section is organized as follows: In Section 3.4.1 we introduce the configuration management system by its meta-model and high-level concepts. In section 3.4.2 we describe the type level product modeling offered by the system in terms of its control plans. In Section 3.4.3
we describe the instance level product structuring mechanisms. In Section 3.4.4 we describe the concepts for creating and controlling product instances.

3.4.1 PCMS meta-model

The company applies the Product Configuration Management System (PCMS). The following description is built on PCMS manuals [114] and experiences gained during process implementation and use.

![Diagram of PCMS Meta-model]

Figure 3.2: PCMS Meta-model

The PCMS meta-model is sketched in Figure 3.2. The meta-model cannot be changed by PCMS users, it is fixed by tool vendors. PCMS privileged users can create a type level model, the PCMS control plan. The control plan can be seen as a customization of PCMS to fit the user organization, bounded by the PCMS meta-model. Unprivileged PCMS users create instance level models, constrained by the control plan and the PCMS meta-model. The entities and relationships in Figure 3.2 are explained in the following two sections.

3.4.2 Control plan

A control plan constitutes the type level model in PCMS. A given product-id has exactly one control plan\(^3\). Control plans can be defined at two levels: Installation and product. A given product may redefine/add-to those given by the PCMS installation.

In the following we describe the PCMS concepts that can be defined in a control plan. Furthermore, since the PCMS manuals do not use a consistent terminology, we describe the terms used in this work. The chosen terminology is the one depicted in Figure 3.2. The following definitions can be made in a control plan:

\(^3\)As we will see later this is a problem with respect to using the PCMS*CTS.
1. **Product Item Type**: Valid types of a *product-item*. PCMS *product items* correspond to Unix files, but PCMS adds functionality such as typing, versioning, and concurrency control. We will hereafter denote a product item as an *item-id*, and a product item type as an *item-type*.

2. **Change Document Type**: Valid types of change documents. PCMS provides two categories of *item-types*, namely *product item types* and *change document types*. The latter is only relevant when the PCMS Change Tracking System (PCMS*CTS*) is applied. PCMS*CTS* was not used by the Company when this case study was performed. PCMS*CTS* is described in Section 6.3.

3. **Rules**: Rules govern the interaction between *change-documents* and *product-items* for change management. Rules are only relevant when the PCMS*CTS* is applied, and are further described in Section 6.3.

4. **Life-cycle**: Each *item-type* must have exactly one *life-cycle*, but a life-cycle can be used by many *item types*. A life-cycle is a state transition diagram, the states being document statuses. Each transition is defined by an initial state, a final state, and an associated role. Users are allocated to roles, and only users having the appropriate role can perform the corresponding transition. When a transition is accomplished, the users connected to the next transition will be notified by e-mail.

5. **Design Part Categories**: Valid types of design-parts. PCMS design-parts correspond to Unix directories, but PCMS adds functionality such as types, variants, and attributes. We will hereafter denote a design part as a *part-id*, and a design part category as a *part-type*. A *part-id* belongs to exactly one *part-type*, but a *part-type* can be used by many *part-ids*. In the definition of a *part-type*, one can specify the intended *item-types* of the *product items* stored in a *part-id* of this type, however, type conformance is not enforced by PCMS.

6. **Roles**: Valid role-titles which may be assigned to members of the project-team and specified in the life-cycle of product items and change documents, to control the access to the product.

There are two PCMS defined *roles* associated to creating and changing control plan contents. The generic control plan is to be made by the PCMS *tool manager* when PCMS is installed. The product specific control plan should be defined by the *product manager* before product development begins.

In the following sections we describe the use of control plan concepts in terms of the part structure, the relationships between parts and items, life-cycles, and product development.

### 3.4.3 Product structure

A PCMS product structure is a logical structure similar to Unix directories. A PCMS installation may contain many products, each being defined in terms of a *product-id*. A *product-id* becomes the root of a hierarchical *part-id* structure.
Part structure

A given part-id may have many children part-ids, but has only one parent part-id (or the product-id). These relationships are called BREAKDOWN relationships in PCMS.

In addition to the BREAKDOWN relationships, part-ids can be related by USAGE relationships. The USAGE relationship corresponds to Unix symbolic links, and gives the possibility to create a Directed Acyclic Graph (DAG). A parent part-id may use a child part-id that belongs to another part-id in the breakdown structure.

PCMS also provides for part-variants. Part-variants have the same part-id, but different variant-ids. Some definitions made to a part-id will be equal for all its variants, e.g. parent, role, and attributes. Other definitions can be different between two variants, e.g. child and usage. A given variant can either be OPEN, CLOSED, or SUSPENDED, but at any time, only one variant can be used (OPEN).

The part structure is defined by means of a PCMS graphical interface, and new parts can be added during product development. A part-id must have a part-type defined in the control plan. This relationship must be established when a part-id is created.

The product manager is a special PCMS role having all possible rights within a given product-id. It is the responsibility of the product manager to create parts. When a part has been created, the product manager may allocate a user to another special PCMS role called the PCMS-part-manager. The PCMS-part-manager have all possible rights within a given part-id. Users allocated to these PCMS roles may allocate other roles and users to the part-id.

When this is done the allocated user is granted the rights of that role on the part-id and all the subordinate part-ids in the BREAKDOWN structure. Any number of roles can be defined, and more than one user can be allocated to each role. A child part-id can redefine the role, overriding the parent definitions. The rights associated to a role is defined by means of life-cycles, this is explained in the item section below.

![Diagram of PCMS Product structure](image)

**Figure 3.3: PCMS Product structure**

**Example 3.1** Figure 3.3 depicts a product BREAKDOWN structure, with product-id S1 as root. S1 has one defined role, namely Designer. One person is allocated to S1’s Designer role, namely John Bull. S1 has two parts, part-id A and B. These two parts are of part-type Sub-system. B defines the Designer role and allocate Bobby Brown to this role. In effect, Bobby Brown, and not John Bull, is the Designer for the part subtree starting from B.
Furthermore, A has two parts, with part-id A1 and A2. B has one part related by the BREAKDOWN relationship (B2), and one part related by the USAGE relationship (A2). The PCMS terminology is A2 being a common design part to A and B. However, users who have roles for the USAGE parent does not inherit any access to the USAGE child part.

With these definitions, John Bull is Designer for S1, A, A1, and A2, while Bobby Brown is the Designer for B and B2. □

Parts and items

An item-id is OWNED by exactly one part-id. A given part-id may OWN many item-ids. This relationship must be established when the item-id is created. As mentioned above, a user must be allocated to at least one role in the part-id to create an item-id OWNED by that part-id.

In addition to the OWNED relationships, item-ids can be related to part-ids by USAGE relationships. While a given item-id is always OWNED by exactly one part-id, it can be USED by other part-ids. However, access rights follows the OWNED relationships, and not the USAGE relationships.

Item life-cycles

A life-cycle is independently defined, but can only be associated to item-types (e.g. not to part-types). A given item-type must have exactly one life-cycle, but a life-cycle can be used by many item-types.

A life-cycle is a state transition diagram defined in the control plan. It is defined in terms of a set of transitions. Each transition is defined by a start state, an end state, and a role. States are user defined strings; these strings will denote legal item-id statuses (of item-types using the life-cycle). The name of the transition must be a role defined in the control plan (more than one is allowed).

One of the start states must be defined as the initial state, and one of the end states must be defined as the final state. Between the initial state and the final state there must be a single path defined as the normal life-cycle. These states and the normal life-cycle has a particular meanings within PCMS. The semantics of the initial state is explained when we discuss item operations in the item section.

![Figure 3.4: PCMS Item lifecycle](image)

**Example 3.2** An example life-cycle is depicted in Figure 3.4. The initial state being OPENED and the final state being ACCEPTED. The normal life-cycle is marked with double lines, it
goes from the OPENED state to the DEFINED state, and ends in the ACCEPTED state.

When an item is created it enters the OPENED state. Only one state transition is possible, to go from OPENED to DEFINED. Only a user allocated to the Designer role can perform this transition. When the item is in the DEFINED state there are two possible next states, ACCEPTED and REJECTED. Only a user allocated to the Reviewer role can perform this transition, this user can choose between the two alternatives. □

PCMS uses roles for two purposes, namely access control and notification. First, only users allocated to the identified role can perform the corresponding transition. Second, when a transition is accomplished, the users connected to the next transition will be notified by e-mail. Furthermore, the item will be added to the user's pending list. The item is removed from the pending list when the required action has been performed.

3.4.4 Creating and controlling products

In this section we describe the version tree, operations that can be performed on specific versions (items), and the relationships between item operations and the item's life-cycle.

Items

An item-id is OWNED by exactly one part-id. An item-id has exactly one item-type. A template file can be associated with each item-type. The use of template-files is to establish an item skeleton with PCMS before work is started on it. An item-type has exactly one life-cycle type.

\[
\begin{aligned}
\text{Item-id} & \quad \rightarrow \quad \text{Item1.1} \quad \rightarrow \quad \text{Item1.2} \quad \rightarrow \quad \text{Item1.3} \quad \rightarrow \quad \text{Item1.4} \\
& \quad \rightarrow \quad \text{Item2.1} \quad \rightarrow \quad \text{Item2.2}
\end{aligned}
\]

Figure 3.5: Version tree: Revisions and variants

Item versions

An item-id is the root of a version tree, containing many revisions and variants. To identify a physically stored file, one must specify the complete PCMS item-spec. The complete item-spec must contain the following: Product-id, item-id, variant, type

\footnote{\textsuperscript{4}} and revision. We have adopted the term item-id to denote the root of a version tree, and the term item to denote a specific version (revision and variant). It is important to note that each item follows the defined life-cycle, e.g. when a new revision is created it enters the initial state of the life-cycle.

\textsuperscript{4}This type refers to the file extension used in the user work-space, this is different from the item-type.
Item modification control

PCMS only permits changes to items by a user who has been assigned both: The responsibility (at least one assigned role) for the part-id that OWNS the item-id, and the role specified for the specific transition in the life-cycle which is being actioned.

There is one exception to this general rule, the product manager is permitted to action\(^5\) any item to any valid life-cycle state.

Item operations

In the PCMS user interface the version tree is shown graphically, the user can select one or more items, and then perform item operations on the selected set. Operations performed on items are closely connected to the item’s life-cycle. The most used operations and their relationship to the life-cycle are as follows:

- **Create item.** The item is created in PCMS\(^6\). When an item is created it always enters the initial state of the corresponding life-cycle. The initial state is different from the other states because it is the only state in which items can be extracted and returned without creating new revisions. The user must have the initial role in the normal life-cycle, and be assigned that role in the owning part-id.

- **Extract item.** An existing item is extracted into a user work-space (Unix directory) for editing. An extracted item is locked for other users until the item is returned by the user. The user must have the initial role in the normal life-cycle, and be assigned that role in the owning part-id.

- **Cancel item update.** Cancels all effects of an extract operation. The item must have been previously extracted by the user who is canceling the update.

- **Return item.** When an extracted item is returned to PCMS, one of two things may happen: If the item is in its initial state, the existing item is overwritten. If the item is in any other state, a new revision of the item is implicitly created, the new item revision enters its initial state. The item must have been previously extracted by the user who is returning it.

- **Action item.** Actioning an item means to perform a transition from the current state to a next state in the life-cycle. The item cannot be currently extracted. The user must have the appropriate role for the current state in the life-cycle of the selected item and be assigned that role in the owning part-id. If these constraints are satisfied, the user selects one of the possible next states.

- **Fetch item.** Takes a copy of the item and places it in the user’s directory. PCMS does not regard the fetched items as being extracted for update. The user must have a role for the owner part-id.

\(^5\)PCMS terminology: To action an item is to perform the PCMS “action item” command on the item.

\(^6\)The create item operation always needs a complete item-spec, we use the term item to denote a unique combination.
• **Browse item.** Calls a read-only editor which allows the user to view a the item’s file content. No changes are allowed on the data. The user must have a role for the owner part-id.

• **Update item attributes.** Updates the values of any of the user-defined attributes for an item revision. User access constraints are explained in the attribute section below.

PCMS also provides other maintenance operations such as to compare items, delete items, relate items to parts, unrelate item from part, and move items to other parts.

![Diagram of PCMS item operations](image)

**Figure 3.6: PCMS item operations**

**Example 3.3** The typical operation sequences are depicted in Figure 3.6. (a) An item is created, then extracted and returned several times while it is in its initial state. Finally it is actioned through its state transitions until one of the final states is reached. If it is extracted when it is not in the initial state, a new item revision is automatically created. (b) An extract operation on an item must be followed by either a return or a cancel operation.

**Item history**

PCMS maintains a history of each item-id. A history entry is created each time the version or status of the item is modified (create and action item). PCMS provides reports showing the history of parts and items.

**Attributes**

Any number of *user defined* attributes can be created on parts and items. Each attribute is either an item attribute or a part attribute. Each attribute can either be defined for all part/item-ids in the product, or only for part/item-ids of a specific part/item-type.

Attribute values must have one of the following four predefined data-types: String, Date, Real, or Integer. Default values can be specified. Furthermore, the attribute can be defined to be updatable or not. If the attribute is not updatable, it will receive the default value or the value given when the part/item is created.

To update an attribute value, the user must fulfill the role requirement, as specified in the attribute’s role check parameter. The following role requirements can be specified:
• **Role.** Values can only be updated by a user who holds this role for that part-id or for the part-id which owns that item-id.

• **Pending.** For an item, values can only be updated by users who have the item in their pending list. For a part, values can only be updated by the **PCMS-part-manager**.

• **None.** Values can be updated by any user who holds any role in the product.

Attributes are *defined* for part/item-ids, but values are *assigned* to specific parts/items. Within an item-id, different *item versions* will have the same attributes, but possibly different values assigned to them\(^7\). Similarly for *part variants* within a part-id.

### 3.5 Company usage of PCMS

In this section we describe the use of PCMS within the Company. By use, we mean product structuring and control plan definitions. The use of PCMS can be regarded as a constraint or a possibility with respect to process modeling. It is a constraint because it limits the choice of product abstractions in the process models, if we want to maintain a correspondence with reality. It is a possibility because PCMS definitions and structures facilitate process elicitation for two reasons, because project participants are familiar with PCMS concepts, and because PCMS itself provides useful information to the modeling (e.g. volume information, history, types, structure).

The section is organized as follows: In Section 3.5.1 we explain that the studied project (PRO1) involved two PCMS products, one for the design structure and one for general project documents. Section 3.5.2 describes the design structure, and Section 3.5.3 describes the document structure. Section 3.5.4 gives an interpretation of the reason for having two product structures. In Section 3.5.5 we describe the Company use of control plan concepts, namely part-types, item-types, life-cycles, and roles. Furthermore, this section describes the relationship between the control plan definitions and product definitions found in the Engineering Handbook.

#### 3.5.1 PCMS products

The studied project was a modification project where a previously developed SDL system was modified to meet new requirements. The old product existed in PCMS as a PCMS product. Hereafter the old product is denoted as OldProduct.

During product development, the Company made new revisions of OldProduct items, they did not create a new product structure for the new system. To be more specific, they used only a specific sub-structure (part-id) of the OldProduct product structure, namely one part structure called *sw design structure*, depicted in Figure 3.7. This part structure only contained the SDL related documents, i.e. those documents that are needed to generate the executable software system. The general project documents were stored in another sub-structure in the OldProduct.

---

\(^7\)There is a PCMS option to force all versions to have the same value.
The project documents of PRO1 were stored in a PCMS product called PROJECT. This product has several sub-structures, one is called PRO-X. The part-id PRO-X denotes the overall context of three related projects, namely PRO1, PRO2, and PRO3. Each of these projects was represented with a part-id owned by PRO-X, and named after the corresponding project. The situation was even more complicated because PRO-X owned other parts that contained documents relevant for all of these projects (e.g. part-id PLANS). However, we studied one of these projects, namely PRO1, and the following discussion is limited to the sub-structure rooted by this part-id, depicted in Figure 3.8.

![Software design structure](image)

Figure 3.7: PCMS parts: Software design structure
3.5.2 The product design structure

The sw design structure is depicted in Figure 3.7. Note that the structure is a part-id structure (instance) and not a part-type structure. However, each of the part-ids has a part-type. The root of this structure is of part-type configuration, the next two levels are of part-type sw-functional-block, and the final level is of part-type sw-module. This structure corresponds to the system breakdown structure in SDL and the associated Structured Design Tool used by Company. In this case the part structure was designed and used according to the PCMS manual. The design parts correspond to SDL design constructs such as system, sub-system, block, and process.

Figure 3.7 depicts the complete part structure, but part names (part-ids) are substituted with anonymous names. This is according to an agreement with the Company not to show real product names of parts and items.

Items and revisions are not depicted in Figure 3.7. However, each part-id owned items of various types, e.g. system parts owned Block Interaction Diagrams (BIDs) depicting the interaction between lower level blocks.

![Diagram of product design structure]

Figure 3.8: PCMS parts: Project document structure

3.5.3 The product document structure

The part-structure of the PRO1 project documents is depicted in Figure 3.8. The root of this structure is of part-type sub-product, the part-ids called Baseline and System Design are of part-type configuration, the remaining parts are either of type sub-product or project
documents. In the latter case I did not find any particular pattern in the part-id typing. Figure 3.8 depicts the complete part-structure, but here I limit my discussion to the quality assurance and the development document parts.

The quality assurance part-ids were organized according to process types, with one child-part for Change Management processes, and one child-part for Document Review processes. These parts contained Change Reports and Review Reports, one document for each process that had been performed within the project. These reports were used in the experiment described in Chapter 7.

The development documents part-ids were mainly organized according to development phases, i.e. analysis, system design, detailed design, implementation, integration, basis software, and system integration. Below these parts there are some lower level parts, organized according to product types (SID, MID) and design structure (SYS-1, SYS-2). However, the rationale for making these parts cannot be explained from a process perspective.

Figure 3.8 illustrates the items that were used in the enactment experiment described in Chapter 7 (page 177). The items are illustrated with their EH product type (e.g. CR - Change Request), and a number showing the number of item-ids (e.g. there were 102 change request documents). The documents are shown in their correct position in the part hierarchy, the part-id shown by a dotted line.

3.5.4 Design and document structures

The Company store their documents according to PCMS part-structures. Some documents are stored according to the design structure of the developed product, and some documents are stored according to the processes that produce them.

The documents stored according to the design structure are the SDT definitions used to generate source code, or source code used to generate executable code. These are the product types that are traditionally targeted by Configuration Management systems, and this is also the case for PCMS: "In the design phase of a project the product is broken down into its logical functions. As the process continues the functions become more and more detailed until eventually a complete design structure of the product is achieved." (Quoted from the PCMS User Manual).

This type of organization is motivated from baselining and system build functionality. E.g. PCMS baselining: "When a baseline is to be created, the user defines how much of the product design structure is to be baselined, by specifying the top-level design part which is to be included." (Quoted from the PCMS User Manual). PCMS assumes that everything is organized according to the product structure, e.g. that a baseline must include every item in the selected part structure. PCMS does not acknowledge orthogonal viewpoints on the product structure, e.g., that one may be interested in documents produced in a certain development phase.

The documents stored according to processes are those that cannot be used as inputs to software tools which generate source code or executable code. These documents describe plans, procedures, meetings, or non-executable software specifications. A major concern in the organization is that such documents can easily be identified according to when they were
produced, as well as the context in which they were produced.

As an example; a Company phase cannot end before every document produced (or intended to be produced) have the PCMS status APPROVED. Since PCMS is organized according to products, it is easier to perform and assess the phase ending criteria if every document is stored according to the phase that produced it.

### 3.5.5 Control plan

In this section we describe how the PCMS control plan concepts were defined and used. The relevant control plan concepts are part-types, item-types, life-cycles, and roles. The description is based on control plan reports, released by the Company for this study. Whenever applicable, we describe the relationship between the control plan definitions and product definitions found in the Engineering Handbook.

#### Part types

PCMS requires that every Part-id must have an associated Part-type\(^8\). The following part-types were defined in the two PCMS products:

- **OldProduct**: The root part-id named Software Design Structure was of the type CONFIGURATION. All other parts were either of the type SW FUNCTIONAL BLOCK or SW MODULE. The first type corresponded to SDL blocks, the second to SDL processes (SDL blocks are decomposed into other blocks or processes).

- **PRO1**: The root part-id named PRO1 is of the type SUB PRODUCT. The lower level part-ids are of the type PROJECT DOCUMENTS, CONFIGURATION, DOCUMENTATION, SUB PRODUCT, or SW MODULE.

The possibility to specify the item-types to be contained in a part-type was not used. If this option had been used we could have checked the types of the item-ids against the part specification. We could then have had an idea of conformance and stability in the part structure.

As used within the Company, part-types only have consequences for the PCMS reports which use the part-type as a selection criterion. However, there are no predefined PCMS reports that are based on part-types, and this is only relevant for user defined reports.

#### Item types

PCMS and EH operated with a different typing system. Product types in PCMS were according to a company standard, while the types used in the EH were according to the terminology used by this particular software development group.

---

\(^8\)Part category in PCMS terminology.
<table>
<thead>
<tr>
<th>PCMS item types</th>
<th>Engineering Handbook product types</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1 - Functional specification</td>
<td>CRS - Customer Requirement Specification</td>
</tr>
<tr>
<td>A2 - Requirements specification</td>
<td>SRS - System Requirement Specification</td>
</tr>
<tr>
<td>A3 - Design specification</td>
<td>BID - Block Interaction Diagram</td>
</tr>
<tr>
<td></td>
<td>MID - Module Interaction Diagram</td>
</tr>
<tr>
<td></td>
<td>ITS - Integration Test specification</td>
</tr>
<tr>
<td></td>
<td>MRS - Module Requirement Specification</td>
</tr>
<tr>
<td></td>
<td>SS - System Structure</td>
</tr>
</tbody>
</table>

Table 3.1: Differences in product typing between PCMS and EH

The differences between the defined PCMS item types and the product types defined in the Engineering Handbook are illustrated in Table 3.1. The differences are substantial, and the PCMS item types have no relevance for the development process. However, there exists a mapping because the EH product types were coded into the item name (the item name was prefixed by the name of the EH product type).

Beside being used as PCMS types the Company standard types are not used for any purpose by the Group. The item type is the input to the PCMS Query item command, which produces item reports (e.g. attribute values, and item history). Thus, the consequence is that the EH types, which are the ones that interest the people in this group, cannot be used in PCMS reporting. Furthermore, the quality assurance activity, which is based on the EH, receives no support from the PCMS query capability. One reason for this situation is that many of the PCMS products were made before the current EH definitions, thus there are part structures and items that depend upon old definitions. However, since control plans can be different for different products, it is possible to evolve the item types from one product to another.

Lifecycle

Most documents had the same PCMS life-cycle, called DOC-1. No document types had a more complicated life-cycle, a few types had a simpler life-cycle. The DOC-1 life-cycle is depicted in Figure 3.9. The normal life-cycle is a sequence of UNDER-WORK, DRAFT, and APPROVED. The other alternative, without a loop, is a sequence of UNDER-WORK, DRAFT, and REJECTED.

![Figure 3.9: The DOC-1 lifecycle](image)

9The only identified usage is that the type is printed on every document, however, this can also be achieved by having the company type as a PCMS attribute.
The transition from DRAFT to UNDER-WORK, which makes a loop in the graph, is a non standard use of PCMS. In this case the item is actioned from a non-initial state to the initial state. The consequence is that the item can be extracted again without creating a new item revision. The product manager can do this operation from all life-cycle states without specifying it in the life-cycle definition. Then, the PCMS life-cycle paradigm breaks down as no new revisions are created and no history entry is made in the item history.

This option was often used in the change management and document review processes. The only motivation was to reduce the number of revisions in order to save disk space. The consequence was loss of information concerning these two processes. In particular, we could not assess process iterations because new updates would overwrite the old information.

Roles and users

In PRO1 every project member (10) was allocated to the author role in the top level part-id (PRO-X), this was not redefined by lower level parts. In effect, every project member could create any document that used this life-cycle, and furthermore change the status to DRAFT.

Only the PRO1 project manager was allocated to the reviewer role. The PCMS product manager role was given to another project participant. In effect, there were two people who could action an item to the other life-cycle states.

The only life-cycles that did not use author as their initial role are the SIMPLE and SW-GENERATED life-cycles. These two life-cycles have only one transition, from the initial state to the final state. The role associated to the transition was the programmer role, 5 people were allocated to this role.

The main problem encountered by the Company personnel about the PCMS role concept was the mix of access rights and notification. Within Company people needed to have access rights on an item without being notified about life-cycle transitions, and furthermore, without having the item entered in their pending list. Because PCMS does not separate between these two concepts, project participants received a lot of e-mails from PCMS, and useful messages tended to drown in noise.

3.6 Summary

This chapter has defined my case to be the software process and product descriptions of a particular project within a particular organizational unit. I have characterized the studied organization, its Engineering Handbook, and the Product Configuration Management System. Furthermore, I have described the use of the Engineering Handbook and PCMS within this organizational unit.

The process descriptions existed as text and diagrams found in the Company's Engineering Handbook. The product descriptions existed both as product descriptions within the Engineering Handbook and as PCMS control plan definitions and product structures.

The main goal of the Company with respect to the EH is to reduce the effort needed for
instantiation, maintenance, and feedback, while maintaining or increasing the EH quality. This is the Company’s motivation for looking into formal process modeling languages and supporting environments.

The following chapter describes a process modeling language, and presents the process models developed on the basis of the information described in this chapter.
Chapter 4

Modeling company processes

The previous chapter described the studied case; the Engineering Handbook and the Product Configuration Management System. My goal is to gain process modeling experiences. To gain experiences I applied a process modeling language to the identified process and product information. This chapter describes the applied modeling environment and the resulting models.

The chapter is organized as follows: Section 4.1 gives an overview of the APEL Version-2 modeling environment. Section 4.2 describes the APEL activity model, its concepts and the graphical conventions used in the modeling activity. Wherever applicable, the APEL modeling concepts are interpreted in terms of PCMS concepts. Section 4.3 describes both the APEL product modeling concepts and the developed product model. Section 4.4 describes both the APEL role modeling concepts and the developed role model. Section 4.5 depicts and describes the APEL model of the System development process. Section 4.6 shows the APEL model of the Document review process. Section 4.7 shows the APEL model of the Change management process.

4.1 The APEL Environment

The APEL v2 tool consists of four editors in which the modeler may describe different aspects of the software process. Each editor offers a different set of concepts and can be used independently from the others. By applying these concepts we create a model of the corresponding aspect. Additionally, each editor provides a possibility to create and browse links between the different models. The set of such models are called project models in APEL terminology. The different APEL editors can shortly be described as follows:

1. **Product editor.** Here we may describe our products in terms of product types and their relationships. Concepts include product types, attributes, methods, inheritance, aggregation, and user defined associations.

2. **Activity editor.** Here we may describe our processes in terms of activities and their relationships. There is no typing system for activities, instead APEL uses variables with
local and global properties\textsuperscript{1}. Concepts include variables that can either be activities, input products, output products, workspaces, roles, data-flows, control-flows, or events. Some of these can be typed according to user defined types (product, workspace, role, team), others by predefined types (data flow, control flow). All concepts may have associated local and global properties.

3. **Role editor.** Here we may describe team and role types and their relationships. Roles can be linked by inheritance. A team can be an aggregate of other teams and/or roles.

4. **Workspace editor.** Here we may describe repository hierarchies, product propagation, and coordination policies. In our case a work-space corresponds to the Unix directory into which PCMS items are extracted. Since neither PCMS nor Unix provides functionality similar to APEL workspaces, and since the studied organization was not familiar with workspace concepts, this APEL aspect was considered as out of scope with respect to our modeling.

As explained in the case study overview (section 3.1) we will not report experiences with respect to the APEL modeling environment, but on the application of modeling language concepts in the context of the studied company. An evaluation of the modeling environment can be found in [57].

### 4.2 The APEL process modeling language

This section explains the modeling constructs used in the developed activity models. The developed models are presented from section 4.5 (page 74) and onwards. Most of the modeling effort was spent in activity modeling, and this is reflected in the following presentation.

Two things should be noted: First, the presented graphical symbols are not exported from the APEL environment, but from an ordinary drawing tool. As a consequence the symbols used are not identical to those used in the APEL environment. Second, some of the semantics of the APEL PML constructs are adapted or interpreted to suit modeling needs. The main differences is that input and output is interpreted in terms of PCMS operations, and that the APEL sequencing semantics is relaxed. In the following I mainly describe the constructs as they are applied in my modeling, commenting on APEL semantics when needed.

This section is organized as follows: Section 4.2.1 presents issues concerning a single activity. Section 4.2.2 presents issues concerning activity chaining. Section 4.2.3 presents issues concerning activity decomposition. The APEL product modeling and role modeling are presented in separate Sections.

#### 4.2.1 The activity concept

By the activity concept we mean issues related to a single node in an activity network or decomposition hierarchy. All activities in the developed APEL models correspond to activities found in the Engineering Handbook.

\textsuperscript{1}A variable and its global properties resembles the type concept.
Figure 4.1 gives an overview of the concepts related to a single activity. The concepts and the applied semantics are as follows:

1. **Control input**: A signal that causes the activity to become active. Nothing happens if the activity is already active.

2. **Enter**: A condition that is evaluated when the activity starts, it might be evaluated to true or false.

3. **Active**: An activity is active from the point in time when it receives a control-input. It is active until some agent decides to terminate the activity.
   (a) **Role**: The personnel allocated to a role may perform work when the activity is active. Externally this is seen as reading input and writing output.
   (b) **Data input**: Can be read as long as the activity is active.
   (c) **Data output**: Can be written as long as the activity is active.

4. **Exit**: A condition that is evaluated when the activity terminates, it might be evaluated to true or false.

5. **Control output**: A signal is sent when the activity terminates.

The following sections describe each of these concepts with respect to how they were used in my modeling. This is organized into roles, control input and output, data input and output, and the enter and exit criteria.

**Roles**

The roles depicted in the activity models represent roles as they are found in the EH, not PCMS roles. The main function of PCMS roles is to control access rights to parts and items. The main function of a EH role is to distribute work among project participants.

The role associated to an activity is a role type representing one or more process agents which performs the work within the activity. If a role occurs two times in the same diagram, it means that a single agent should perform the associated activity within the same activity instance.
Control input and output

By control input we mean how an activity reacts when it receives a control signal. In APEL, an input signal causes the enter criteria to be evaluated, if satisfied the activity becomes active. If not satisfied, the enter criteria is periodically evaluated until it becomes true. Only the first control signal has any effect.

This kind of control semantics assume that it is possible to state the exact enter criterion. In our case this was not possible. Enter criteria were always idealized, and the starting of activities could not be directly related to such conditions. Decisions about starting and stopping activities were human decisions. Thus, our control inputs must be read as “some human decides to start the activity”. At this point the enter criteria might be true or false, but the activity is started anyway.

By control output we mean how and when an activity sends a control signal. An APEL exit criterion is a verification criterion, evaluated when the activity terminates. If false, the activity terminates with failure, else it terminates with success. I apply the same meaning to the exit criterion, but prefer to denote it as terminating with a true or false exit criterion.

Data input and output

An activity represents a transformation of inputs into outputs by means of human resources. Every activity input and output is a product type, defined in the product model. Each of the product types corresponds to a document type identified in the Engineering Handbook. A document type in the input and output lists denotes an unknown number of document instances of this type (PCMS item revisions).

<table>
<thead>
<tr>
<th>Files:</th>
<th>Paper copies:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activity name</td>
<td>Activity name</td>
</tr>
<tr>
<td>DOC1</td>
<td>DOC1</td>
</tr>
<tr>
<td>DOC2</td>
<td>DOC1</td>
</tr>
<tr>
<td>DOC3</td>
<td>DOC2</td>
</tr>
<tr>
<td>DOC6*</td>
<td>DOC6*</td>
</tr>
<tr>
<td>DOC7(C=N)</td>
<td>---</td>
</tr>
<tr>
<td>DOC8(Attribute=Value)</td>
<td>---</td>
</tr>
</tbody>
</table>

Figure 4.2: Transformation of input into output

The relationship between input and output types is depicted in Figure 4.2. The interpretation in terms of PCMS operations are as follows:

- **Read only.** A product that appears in the input list but not in the output list. This corresponds to PCMS “fetch” or “browse” item commands, possibly on all items of the specified item-type.

- **Create.** A product that appears in the output list but not in the input list. This corresponds to the PCMS “create” item command.
• **Update.** A product that appears in both lists. If the item is in its initial state, the PCMS interpretation is to create a new item revision, whereas an item in its initial state is overwritten. This corresponds to pairs of PCMS “extract” and “return” item commands.

The relationship between configuration items and document paper copies is shown with a “mark” on the document type (DOC'). The input/output relationships are as follows:

- **DOC, DOC’** Means that DOK is printed in that activity.
- **DOC’, DOC’** Means that the paper copy (DOC’) is annotated in that activity.
- **DOC’, DOC** Means that the configuration item is updated according to the paper copy.

Initially I modeled attributes and values in the activity model, but these were removed because they make the diagram difficult to read. If the control flows are abandoned, showing attributes can be acceptable. Generally I adopted the DOK (Attribute=Value) convention, but simplified into DOK(Value) when referring to PCMS life-cycle states. If the document attributes or life-cycle is to be updated, the possible values are shown both in the input and output list. Cardinality can be described with the DOC(C=N) convention, meaning max N input or output documents of type DOC.

### 4.2.2 Activity chaining

In the previous section we described the semantics of individual activities. In this section we describe the chaining semantics in terms of control flow, data flow, and events. An example process which applies these concepts is depicted in Figure 4.3.

![Diagram](image)

**Figure 4.3: Example: Activity chaining**

**Example:** The control-flow in Figure 4.3 starts with a small black triangle, the control flow then enters an and-out symbol. From the and-out symbol there are two paths, one flow to
activity-3, and one flow to an or-in symbol. The other in-signal to the or-in symbol originates from an event.

Example: Each of the data-flows in Figure 4.3 starts with the product-reference symbol, showing that the input products are not created in this process. From the product reference symbols there are data-flows to activity inputs. Furthermore, there are data-flows from activity outputs to the inputs to other activities, or to output reference symbols.

Control flow

By control flow we mean how a control signal is transported from the sender to possibly many receivers. How control is transferred from one activity to another is specified by use of control flow connectors.

![Control flow connectors](image)

Figure 4.4: Control flow connectors

The possible control flow connectors are depicted in Figure 4.4, their meaning is as follows:

- **OR-IN** The control out-signal is sent when one control in-signal arrives.
- **OR-OUT** When an in-signal arrives one or more out-signals are sent, according to an associated condition.
- **AND-IN** All in-signals must arrive before the out-signal is sent. There is a wait until every in-signal has arrived.
- **AND-OUT** When an in-signal arrives, all out-signals are sent at once.

The APEL control-flow assumes a finish-start dependency between activities with producer-consumer relationships. The real constraint in the “case” is often the finish-finish dependency, the consumer cannot finish until the producer has finished. In APEL this can only be modeled by events between parallel activities. Attempting to model this by correct APEL semantics, diagrams becomes unreadable and not representing the understanding of the process performers.

Consequently, I modeled finish-finish dependencies as if they were finish-start dependencies. Furthermore, since the control flow cannot be read in a strict way, one must always keep in mind that the start and stop of an activity is a project manager decision.
Data flow

A data-flow represents a *transportation* of outputs to inputs. The simple case is that one activity (the producer) is responsible for the transportation of one document type to another activity (the consumer). In this case, the producer output type is connected to the same type in the consumer’s input list. In general, the transportation of a document type means to transport every produced document (instance) to the consumer. It is up to the consumer to select among the documents, e.g., to select only the latest item revisions and/or only those with a certain life-cycle state. However, if the consumer specifies a selection criteria, the producer may choose to transport only the documents which satisfy the criteria.

The complex cases are when the output type of a producer needs to be connected to several consumers, or when several producers need to be connected to a single consumer. These cases are complex because there might be a wide range of distribution alternatives. I used a simple approach in these cases, namely a simple split and merge of data flows. In our models, a *data flow* is always used to denote either a *single document type* or an *aggregated document type*. The document aggregates are not important with respect to model understanding. Document aggregates are only used to reduce the number of data flows by grouping flows under a common name. An aggregated flow can always be substituted by the flow of its components.

In the case of *single document type* the split and merge will always have exactly the same type before and after the operation. The interpretation is always related to instances of that type, however, no particular semantics are depicted in the diagrams. In the case of one producer and several consumers the main alternatives are to make all documents available to all consumers, to distribute document sub-sets to each of the consumers, or to give out paper copies to some or all of the consumers.

Events

Figure 4.3 depicts one event-out and one event-in symbol, in and out depicted as arrows. Events are always connected to and or or symbols. The process will typically continue its execution after sending an event. An event does not itself cause any action, actions happen when other processes listen for events.

In my modeling I have used events in a sloppy way. Event-out is only used to indicate the typical starting point of Change management processes. Event-in is only used to indicate the activities that are performed within the Change Management process.

**Example:** The event-out symbol depicted in Figure 4.3 is sent to start a Change Management process. The decision about whether to send the event is associated to an or-out control flow. The event is sent if one or more of the SB and ADTSC documents have the PCMS status REJECTED. The event-in symbol indicates that the subsequent process steps are performed within the the Change Management process.
4.2.3 Multiplicity and decomposition

The symbols for multiplicity and decomposition is depicted in Figure 4.5. Decomposition (fat lines) means that there exists another diagram showing the activity at a decomposed level. Inputs and outputs are shown with the object reference symbol at the decomposed level. E.g. the object reference depicts the input type in one place, and may then flow to any of the decomposed activities. Control flow starts from the control flow symbol (black triangle) as soon as the parent activity becomes active. The parent activity is active as long as one of its children is active.

![Symbols for Multiplicity and Decomposition](image)

Figure 4.5: Multiplicity and Decomposition

Multiplicity means that there might be more than one activity instance. A multiple-terminal symbol means that there are more than one leaf-level activity instances. A multiple-decomposed symbol means that there are more than one instance of the overall activity decomposition.

- With a single activity there is only one activity of this type within the project. Several agents may work on the activity, but it is impossible to distinguish their work in terms of the activity concept. I.e., there is only one start time and one end time, and all work on the same set of inputs and outputs.

- With multiple activities one can allocate different resources to different activity instances. Activity instances may start and stop independently of each other, but the overall activity does not end before every activity instance has ended. Furthermore, activity inputs can be split into sub-sets, and different activity instances may work on different sub-sets.

Multiple activities cause additional needs with respect to the binding of inputs, outputs, and agents. With a single activity one only needs to bind inputs, outputs, and agents to the activity, with multiple activities one also needs to bind this information to individual activity instances. However, such bindings are likely to occur when the process is executing, furthermore, this binding depends upon the execution history (e.g. documents produced, availability of resources, project priorities). This can be hardly be modeled at the type level. At the instance level there are two cases, a single activity instance followed by multiple activity instances (one–many), and multiple activity instances followed by multiple activity instances (many–many). For each document type, the associated set of document instances must be distributed according to some decision made by a human.
In my modeling I use both single and multiple activities. Their usage illustrates the typical situation. In general, there might be more than one activity instance also for the activities modeled with the *single activity* symbol. Because of this “uncertainty” in the choice between single and multiple activities I do not associate them with different semantics, e.g., instantiation or instance-level binding. In the case of multiple activities, one can read the activities as if all were started at the same time, all receive the same input, and all agents can work on all instances. However, one must keep in mind that this does not reflect reality, as the project manager will define strategies when he instantiates individual activities. He will then allocate specific agents to specific sets of documents.
4.3 The APEL product model

APEL supports product modeling according to object-oriented principles. The notation applied by APEL is the one defined by Rumbaugh [105]. There is a single product model within an APEL project model, the product model is presented as a single diagram. The entities and relationships in APEL product models represent product types and relationships among product types.

The two modeled structures are the inheritance structure and the aggregate structure. The inheritance structure is not (and cannot be) reflected in the APEL activity models. The aggregate structure is reflected in the process models through product split and join. The product type denoting the collection in these operations should be defined as an aggregate.

4.3.1 Properties

Boxes in the modeling sheet represents product types, each defined by a unique name. Each product type can be associated with a range of properties, e.g., triggers and actions. However, most of these properties are related to product workspace behaviour, and since workspaces are not modeled, these concepts are not explained.

The modeler may define one or more attributes on each product type. Each attribute can be described by its data type, constraint, cardinality, and default value. In my modeling I chose to model PCMS life-cycles as product attributes. Furthermore, I modeled the status attributes of the Change and Review reports as a product attribute. None of the other APEL properties were applied.

4.3.2 Inheritance

Product types can be organized into one or more sub-typing hierarchies (“is-a”). But, a given product type cannot be a sub-type of more than one super-type (single inheritance). A sub-type will inherit all the properties, aggregates, and associations of its super-type. A sub-type can add, but not redefine properties, aggregates, and associations defined in the super-type.

In my modeling I chose the hierarchical inheritance structure depicted in Figure 4.6. The boxes denote product types defined to be of kind concept. The arrows denote inheritance relationships. The abbreviations below the boxes denote product types defined to be of the kind file. These product types inherit from the product type (box) in which they are placed below. The only reason for not depicting these product types with boxes and lines, is to be able to present the complete structure in one page.

The inheritance structure in Figure 4.6 constitutes a classification of all the product types defined in the Engineering Handbook. Each of the boxes in this figure represents an “abstract” product type (not files). The classification can be described by describing these “abstract” product types:

1. ARTIFACT. The most general concept, it is used to denote everything that we wish to look upon as a product.
2. **PHYSICAL.** Some form of physical device not considered to be a document.

3. **DOCUMENT.** Readable information.

4. **PCMS ITEM.** Documents that reside within PCMS.

5. **NOT PCMS.** Documents that do not reside within PCMS, e.g. Customer Requirements.


7. **PRODUCT.** Ordinary PCMS documents. Documents that exist as ordinary PCMS items, these documents can be extracted, edited, and returned to PCMS.

8. **DOC.** Administrative documents. They do not directly describe an aspect of the physical product.

9. **SWDOC.** SW Document. Describes some aspect of the systems software.

10. **HWDOC.** HW Document. Describes some aspect of the systems hardware.

11. **SWHWDODC.** SW and HW Document. Describes some aspect of the system common to both hardware and software.

The classification of PCMS documents into DOC, SWDOC, HWDOC, and SWHWDODC existed in the Engineering Handbook.

The RR and CR documents are placed under the PROCESS category. This is not true with respect to the current practice within the Company. As of today these documents exist as normal PCMS items. They are placed under the PROCESS category to show the type of documents that are intended by PCMS to be Change Documents. But, as described in Chapter 7, the change and review reports were implemented as PCMS change documents in the context of this thesis, and in this setting the PROCESS category is the correct position in the hierarchy.

The sematical implications of the APEL inheritance structure are threefold: First, the classification can be structured by means of the inheritance tree. Second, all document types that could be used as inputs to the Change management and Document review processes could have a common super-type (DOC). Third, attributes used by all these documents could be defined one place, e.g. author and editor. Document variables in the activity model can reference these types. The visual implications of APEL is that a graphical overview of product concepts might support product understanding.
4.3.3 Aggregation

Product types can be organized into one or more aggregate hierarchies (“part-of”). A given aggregate has one parent product type (“the aggregate”) and one or more children product type(s). A given product type can be part-of different aggregates. The relationships between the parent and each of the children may have different multiplicity, the alternatives are: zero-or-one, exactly-one, zero-or-many, and one-or-many.

In the APEL modeling activity the aggregate structure depicted in Figure 4.7 was modeled. Here, the top level aggregate is PRODUCT, which is an aggregate of six lower level aggregates (PINDOC, ANADOC, STPDOC, HWDDOC, SWDDOC, OTHERS). Each of the six aggregates is an aggregate of product types. The product types are shown as abbreviations, below the corresponding aggregate.

The purpose of these aggregates is to reduce the visual complexity in the top level process model (depicted in Figure 4.9, page 75). As seen from Figure 4.9, each aggregate is the output of an activity: PRODOC (Project), PINDOC (Project Invocation phase), ANADOC (Analysis phase), STPDOC (System Test Preparations phase), HWDDOC (Hardware development activity), SWDDOC (Software development activity), OTHERS (Other activities).
The intended type level semantics are that each aggregate lists the document types produced in the corresponding activity. In PCMS one can only define one level aggregates, this by defining that a part-type should include a list of item-types. A part-type cannot specify other part-types, thus the PROJECT aggregate cannot be expressed in PCMS.

The intended instance level semantics are that each aggregate instance, at any time, includes all PCMS item revisions produced by the associated activity instance (task). If this were to be reflected in PCMS one needs to create a part-id for each task, and place all items produced by the task in this part-id.

![Diagram of Product Aggregates](image)

Figure 4.7: Product aggregates.

### 4.3.4 Associations

In addition to the predefined “is-a” and “part-of” relationships, the modeler may create his own relationships, denoted as associations. Only binary relationships can be created, and each relationship must have a direction (origin and destination). When a relationship has been defined it can be used to relate product types in the same way as for the predefined relationships.

An association is described by means of the following properties: Label, domain, cardinality, and DAG. Label is the name of the relationship. Domain is a specification of the product types that can be origins and destinations of the relationship. Cardinality can be either one-one, one-many, many-one, or many-many. DAG specifies whether or not Directed Acyclic Graphs can be created with the relationship.

We did not model any associations because the Engineering Handbook did not describe general document dependencies (in any other way than through activity input and output dependencies).
4.4 The APEL role model

There is a single role model within an APEL project model, the role model is presented as a single diagram. The entities in an APEL role model represents role types and team types. The relationships in an APEL role models represents “is-a” links among roles, “part-of” links among teams, and “part-of” links between teams and roles.

In an APEL role model, every team and role type is globally defined within the project model. The connection to the activity model is that any activity, at any level of decomposition, can define a role or team variable of a type found in the role model.

- **Role types.** Role objects (hexagons) in the modeling sheet represents role types, each defined by a unique name. Each role type can be associated with attributes and actions defined by the modeler. Attributes and actions for role types are created in the same way as for product types. Role types can be organized into a sub-typing hierarchy through the inheritance relationship (“is-a”).

- **Team types.** Team objects (boxes) in the modeling sheet represent team types, each defined by a unique name. Similarly to role types, team types can also be associated with attributes and actions. Team types can be structured into aggregates through the “part-of” relationship. A team type can be an aggregate of other team types, of role types, or a combination of team and role types.

The developed APEL role model is depicted in Figure 4.8. In this figure team types are denoted by rectangles, and role types by hexagons. The aggregate structure has the project team as root, and contains sub-teams and roles. We found no situation in the Engineering Handbook where it was natural to use role inheritance. We applied the role editor to create role and team types, and an aggregate (“part-of”) structure among these types.

We identified three sub-teams, the Analysis team, the Design team, and the Test team. Each of these teams has a team leader, and an unknown number of team members. In Figure 4.8 the team leader is represented by a single role, denoted as the team responsible. Team members are represented by a single role within each team (analyzer, designer, and tester). The Test team had an additional identified role, namely the Test specifier.

Cardinality is not depicted in Figure 4.8, but APEL provides the possibility to add cardinality information on the aggregation according to Rumbaugh OMT notation [105]. E.g. that a project has one or more implementation teams, and that an implementation team has one or more implementor roles.

72
Figure 4.8: APIC: Teams and roles
4.5 Systems development

This section presents the overall system development process. The studied project (PRO1) did not involve hardware development, and hardware development processes are not described or discussed in the following presentation.

4.5.1 Documents

The input and output documents depicted in the system development process are the leaf-level product types depicted in Figure 4.6 (the inheritance structure), and the aggregated product types depicted in Figure 4.7.

4.5.2 Roles

The following roles are referenced within the system development activity model:

1. Project responsible. A role played by an agent who is not part of the project team, but dependent upon the project in the sense that if there is a project there is also a project responsible.

2. Method responsible. A role played by an agent who is not part of the project team.

3. Project manager. Responsible for the project in general, furthermore, the project manager has a particular responsibility for customer relations.

4. Analysis team. The group of people that do analysis.

5. Design team. The group of people that do system design and detailed design.


7. Implementation team. The group of people that do implementation.

8. Test team. The group of people that do testing.

9. Test specifier. A role within the Test team.

The role as method responsible is independent of any particular project, the others are relative to the PRO1 project.

4.5.3 Activity model

The generic systems development process is depicted in Figure 4.9. The development phases depicted in this figure are Project invocation, Analysis, System test preparations, System test, and Release. These phases are common to both HW and SW. The other four phases, named System design, Detailed design, Implementation, and Integration are different for HW and SW development. These phases are embedded in the Hardware development and Software
development activities in Figure 4.9. Three of the activity boxes in Figure 4.9 are shown with bold rectangles, this denotes decomposition, and means that we show a decomposed diagram in the following sections.

![Diagram](image)

**Figure 4.9: Systems development**

In the following we describe each of the process steps in Figure 4.9:

1. **Project invocation.** Decomposed. The main activity is to go through the contract, to choose development strategy, and to establish and plan the project.

2. **Quality management.** Quality assurance personnel perform a check at phase endings and produce a Phase Acceptance Form (PAF). QA personnel are normally not part of the project team. QA personnel is responsible for methods in general, most notably process modeling (GEH) and process improvement (QIP/GQM).

3. **Project management.** The Project Manager is responsible for customer relationships, managing project resources, managing project methods and terminology, monthly reporting (SR), and for writing the final report (FR). All input documents are possibly
subject to changes by the Project manager, the project agreement (PA) is only changed in agreement with the customer.

4. **Analysis.** Decomposed. The main activity is to analyze and formalize hardware and software requirements.

5. **Hardware development.** Develop hardware (HW) and associated documentation (HWDOC). Outside the scope of the study.

6. **System test preparation.** System test preparations involve the development of both hardware and software for the purpose of testing the integrated hardware and software product. System test preparation is run in parallel with the product development.

7. **Software development.** Decomposed. Develop the software and its documentation (SWDOC). The source code and executable code being part of SWDOC.

8. **System test.** Produce executable code (EC) from source code (SC). Perform system testing on hardware (HW) with executable code (EC). The total system (HW and EC) is placed in a physical test configuration (PTC). Then the testing is performed according to the test plan (TP) by means of system test instructions (STI) that produce the system test protocol (STP).

9. **Release.** Perform customer acceptance testing on the hardware (HW) with executable code (EC). The system (HW and EC) is placed in a physical test configuration (PTC). Then the testing is performed according to the test plan (TP) by means of acceptance test instructions (ATI) that produce the acceptance test protocol (ATP).

The following sections present the decomposed activities.
4.5.4 Project invocation

In this phase the project manager is to re-calculate the project estimate and make the different project plans. The project organization is established, the contract reviewed and analyzed, and the requirements broken down to lower level requirements. Only a few people are involved in this phase, an appointed project manager, a project responsible within the organization, and a method responsible. Of these, only the project manager is part of the project team.

![Diagram of project invocation](image)

**Figure 4.10: Project invocation**

In the following we describe each of the process steps in Figure 4.10:

1. **Contract review.** The Customer Requirement Specification (CRS) is made using a requirement tracking tool. A review team appointed by the project responsible performs a walk-through and writes a Contract Review Report (CRR). The customer is to approve the RTM representation of the requirements.

2. **Make project agreement.** Make a contract between the project manager and the project responsible.

3. **Choose models.** Decide on methods to be used and measurements to be collected according to the PERFECT Improvement Approach (PIA). Implements the PIA characterization and planning activities.

4. **Make project plans.** Make initial project plan (PP), activity plan (AP), and project dictionary (PD).
4.5.5 Analysis

The main purpose of the analysis phase is to produce the System Requirement Specification and System Interaction Diagrams. These documents contain a formalization of the Customer Requirement Specification.

![Diagram of Analysis Process]

Figure 4.11: Analysis

In the following each of the process steps in Figure 4.11 is described:

1. **Functional analysis.** Decomposed. Identify and formalize functional requirements into a System Requirement Specification (SRS) and a number of System Interaction Diagrams (SID).

2. **Implementation analysis.** Risk analysis to identify uncertain or critical areas. Also used to assess the amount of work needed to implement important functions. Produce the Implementation Analysis Report (IAR).

3. **Develop test plan.** Write a description of what, when, and how to perform different types of testing.
The functional analysis activity in Figure 4.11, is decomposed into the diagram depicted in Figure 4.12. In the following we describe each of the process steps in Figure 4.12:

1. Define system boundary and user types. Identify and define the different user types, sub-systems and interfaces. Documented as a section in the SRS.

2. Identify and define functions. The functionality of the systems are grouped and described by defining different functions in the SRS. Requirements to new or changed basis software functions are to be identified and defined. Functions are documented as sections in the SRS.

3. Make SIDs, Message Sequence Diagrams that depict the overall communication between the system(s) and its environment. The purpose is to give an overview of the system, and later to define system test cases.

4. Make functional overview. Shows important pre- and post conditions used in the SIDs. The conditions are defined in a state transition diagram where each condition defines a state. Documented as a section in the SRS. SIDs which change these states are to be defined as signals in the state transition diagram.
4.5.6 Software development

Software development is normally performed in parallel with system test preparations and hardware development. The studied project was a pure software development project, and software development represented the total development effort.

![Diagram of software development process](image)

Figure 4.13: Software development

The *software development* activity in Figure 4.9 (page 75) is decomposed into the diagram depicted in Figure 4.13. In the following we describe each of the process steps in Figure 4.13:

1. **System design.** Decomposed. Decompose the system into blocks and modules, defining the SDL static structure.

2. **Detailed design and implementation.** Decomposed. Define the system behavior by means of state transition diagrams, procedures, and abstract data types. Generate code and perform unit testing.

3. **Integration.** Decomposed. Integrate behavior into blocks and systems while performing tests.
4.5.7 System design

The main activity of the System design phase is to make the block and module decomposition (static structure) and to develop the associated interaction diagrams. Input documents are the System Requirement Specification and a number of System Interaction Diagrams developed in the analysis phase.

The *system design* activity in Figure 4.13 (page 80) is decomposed into the diagram depicted in Figure 4.14. In the following we describe each of the process steps in Figure 4.14:

1. **Define the system in SDT.** This is the first time the SDL tool is used in the development process. Create the SDL system file (<sysName>.sys). The SDL System Structure (SS) is an aggregate of system, block, and process files.

2. **Decompose the system into blocks.** Write a document that divides the system into blocks and states the requirements for each block. In this project both module and block requirements were put into one document, namely the Module Requirement Specification (MRS).

3. **Make BIDs.** Make a number of message sequence diagrams depicting interactions between blocks. Each BID is made by means of the SDL tool, and each is stored as a separate PCMS item.

4. **Finalize the system decomposition.** The blocks are now defined in the SDL tool, creating a number of block files (<blockName>.sbk), stored as separate PCMS items. In this process, the corresponding block requirements may be updated in the MRS document.

5. **Decompose blocks into modules.** Blocks are decomposed into modules in parallel. A sub-section for each module (SDL process) is created in the MRS and the block is decomposed into SDL processes.

6. **Make MIDs.** When all blocks are decomposed into modules, the SDL process interaction for each block is defined and depicted as Module Interaction Diagrams (MID).

7. **Make ITSs.** When all MIDs are finished the Integration Test Specification (ITS) is made on the basis of Block and Module Interaction diagrams. The ITS is a single document describing integration order, how to choose test cases, how to execute tests, etc.

8. **Finalize block decomposition.** When all MIDs are finished the processes are defined in the SDL tool. The definitions are stored in the corresponding block files. The corresponding module requirements may be updated in the MRS document.

The output of this phase is a single MRS document\(^2\), a single ITS document, a number of BID and MID diagrams, one SDL system file, and a number of SDL block files that constitute the complete SDL static structure (SS).

\(^2\)In the three projects BRS and MRS has been put together in one document, i.e., the content of BRS moved into the MRS document.
Figure 4.14: System design phase
4.5.8 Detailed design and implementation

The main purpose of the phase is to transform MRS and MIDs into tested and APPROVED module source code, in SDL terms it is to add system behavior (SB) to the static structure (SS). Source code (SC) is generated by the SDT tool on the basis of SB, SS, and ADTSC files (generates one C-file).

![Diagram of detailed design and implementation](image_url)

**Figure 4.15: Detailed design and implementation**

The *detailed design and implementation* activity in Figure 4.13 (page 80) is decomposed into the diagram depicted in Figure 4.15. In the following each of the process steps in Figure 4.15 is described:

1. **Design module skeleton.** A module skeleton is an SDL state transition diagram without procedure calls; it contains states, inputs, outputs, and textual description of actions. A module skeleton is documented both as an SDL description (SB) and in a Word-processor document (FO). The FO is updated to reflect changes in SB throughout the development process.

2. **Specify ADTs.** Based on MRS and SB, make a single ADT specification document for all ADTs in the system. Specify tests for individual ADTs in a single Module Test Specification document (MTS). For each ADT, at least one SDL process (SB) is modified for test purposes.

3. **Complete and verify module.** The main purpose of the activity is to transform the module skeleton into complete System Behavior. This involves to complement the process graph with error cases, decide on data representation, define local data types and variables, identify the need of ADTs, and complete signal definitions with parameters. The FO document is continuously updated to reflect the actual SB developed. The “Complete and verify module” must be coordinated with the “Specify ADT” activity.
4. **Dynamic analysis.** Performed when every SDL process is completed. Utilizes the functionality of the SDT analyzer for “dynamic semantics”. Upon detection of errors the modules are updated by means of the Change Management process.

5. **Implement and test modules.** Write C-Code for ADTs (ADTSC). Generate code from SB and SS into SC, ADTSC is included automatically. Make Module Test Instructions and perform unit testing, the unit test results are documented in the Module Test Protocol (MTP). Testing is repeated until all tests are approved.

![Diagram](image.png)

*Figure 4.16: Design module skeleton*

**Design module skeleton**

The *design module skeleton* activity in Figure 4.15 (page 83) is decomposed into the diagram depicted in Figure 4.16. In the following each of the process steps in Figure 4.16 is described:

1. **Make skeleton and FO.** A module skeleton is an SDL state transition diagram without procedure calls; it contains states, inputs, outputs, and textual description of actions. It is documented both in SDL (SB) and in a separate Applix Word document (FO).

2. **Analyze skeleton behavior.** The skeleton process is analyzed using the SDL analyzer. Check that states and transitions in the FO are the same as those in the skeleton process.

3. **Verify skeleton and FO.** Check that all MIDs can “executed” by the skeleton process (and FO) and that the same signals and parameters are used.

![Diagram](image.png)

*Figure 4.17: Specify ADTs*
Specify ADTs

The *specify ADTs* activity in Figure 4.15 (page 83) is decomposed into the diagram depicted in Figure 4.17. In the following each of the process steps in Figure 4.17 is described:

1. **Make ADTS.** Each ADT and each of its operations has a separate section in the ADTS. For each operation an interface and a body description (informal) is made. Review the ADT specification.

2. **Make MTS.** The module test normally consists of testing the ADTs. The project may also find it necessary to test SDL process graphs. For each operation, make tests with normal, border, and invalid data input.

3. **Prepare module test.** For each ADT, at least one SDL process (SB) is modified for test purposes. It might be a single process for all ADTs.

![Diagram of Complete and verify module process](image)

*Figure 4.18: Complete and verify module*

Complete and verify module

The *complete and verify module* activity in Figure 4.15 (page 83) is decomposed into the diagram depicted in Figure 4.18. In the following each of the process steps in Figure 4.18 is described:

1. **Complete process graph:** Extend the module skeleton by formalizing decision and action symbols and defining data types and variables. Complement the process graph with error cases. Identify the need of ADTs, and complete signal definitions with parameters. The FO document is continuously updated to reflect the actual SB developed. The “Complete and verify module” must be coordinated with the “Specify ADT” activity.

2. **Verify process:** The completed process graph is verified against the MRS/MID/ADTS. Corresponds to an ordinary review process, but is different in that verification is an explicit check against external documents while review is checking documents for internal consistency.
Implement and test modules

The implement and test modules activity in Figure 4.15 (page 83) is decomposed into the diagram depicted in Figure 4.19. In the following each of the process steps in Figure 4.19 is described:

1. **Implement ADTs**: Write C-Code for ADT’s (ADTSC) based on ADT Specifications (ADTS). Each C-file is stored as a separate PCMS item.

2. **Generate code**: Use the SDL code generator to generate Source Code (SC). The source code is generated on the basis of SS, SB, and ADTSC files. A single SC file is produced and stored as a PCMS item.

3. **Make module test instructions**: Write unit tests for ADT’s. A number of MTI files are produced and stored as PCMS items.

4. **Perform module test**: Make test scripts to run the various MTI files, record the result in MTP files. The test is repeated until all tests are approved.

The Change Management process represents a modeling problem. The problem is that “Implement and test modules” is partly repeated within the Change Management process, and not that the Change Management process initiates the “Implement ADTs” process.
4.5.9 Integration

The integration activity in Figure 4.13 (page 80) is decomposed into the diagram depicted in Figure 4.20. In the following each of the process steps in Figure 4.20 is described:

1. **Make BTIs and test block.** One process for each block in the system. A process may start as soon as the MID document(s) for that block exist (APPROVED). The overall process finishes when every BTP file has been produced and APPROVED for all blocks.

2. **Make ITIs.** Produce ITIs based on BIDs. Try to cover all BIDs, and also consider signal sequences not covered by any BID. Always at least one Review process.

3. **Integration test.** Similar to Block Test, but now on the system level.

4. **Verify ITP.** Similar to Verify BTP.

I have chosen to model the invocation of the Change Management process as an EVENT. A generated event will cause the CM process to start. In the “Make BTIs and test block” process, the block test may start because of a CM process termination.

---

![Figure 4.20: Integration](image)

![Figure 4.21: Make BTIs and test block](image)

87
Make BTIs and test block

The *make BTIs and test block* activity in Figure 4.20 (page 87) is decomposed into the diagram depicted in Figure 4.21. In the following each of the process steps in Figure 4.21 is described:

1. **Make BTIs for block.** The process may start as soon as every MID for this block exists (APPROVED). The ITS document must be APPROVED. The test team write test instructions using SDT simulator commands based on message sequence charts defined in MID. ITS define guidelines. The process finishes when all BTI files for this block exist with status APPROVED.

2. **Block test.** The process may start as soon as every BTI file (script) for that block exists (FINISHED). The process finishes when the test protocol (BTP = BTDMP files + BTLOG files) is generated for that block. Serious errors initiate the Change Management process to change SB.

3. **Verify BTP.** All BTP’s for this block are checked in one “batch” by the test responsible. Initiate a CM process if errors are found.

BTI and BTP obtain the APPROVED status when the block test finishes and triggers the Integration test (a PCMS baseline is taken). This is not the case when the block test is interrupted by a CM process. ITI and ITP obtain the APPROVED status when the Integration phase finishes.
4.6 Document review

During the software development process, process agents are allocated to activities (or activity instances). Within these activities the agent is responsible for producing one or more documents (item revisions), of one or more item-types. If the items already exist within PCMS, the agent extracts these items. If one or more items do not exist, the agent first creates the new items, and then extracts them. In both cases new item revisions are created, and each item revision enters its initial lifecycle state, UNDER-WORK.

The agent then works on his documents until he is finished, or until he wants feedback on the produced documents. The agent then initiates a Document review process. Several authors may decide to have their documents reviewed in one process, in this case one of the authors initiate the review process. There might be several outcomes of a document review process, these are discussed below. Every project document associated with the DOC-1 lifecycle must go through at least one document review process before it can be approved. Documents are not approved in the document review process itself, the approval is done by the project manager in a separate process.

In the following the Document review process is described in great detail. First, the involved documents, the Review Report (RR) and the Reviewed documents are described. Second, the roles are described. Finally, the developed models are depicted and explained.

4.6.1 Documents

The Document review process involves two types of documents, the Review Report (RR) and the documents to be reviewed (hereafter called DOC). For each process instance there are exactly one RR, but possibly many DOC’s. Note that DOC denotes any PCMS item, of any PCMS item-type. The RR keeps process information and will be updated as the process moves forward, e.g. every reviewed DOC is referenced in the RR.

Within the Company, each Review Report is implemented as an ordinary PCMS item, not as a PCMS Change document (explained in Chapter 6). For each project, all RR’s are stored under a common PCMS part, and are named with “RR” plus a sequence number local to the project (e.g. RR-21). In the studied project there were 76 such items. The items have PCMS type C3 (QA-archive) and the DOC-1 lifecycle.

After executing an instance of the document review process, the corresponding Review Report contains the following information:

1. Participants: The agents who should have been present at the review meeting. This corresponds to the agents who participated in the review process.

2. Meeting manager: A process role. The agent who was responsible for the review meeting.

3. Secretary: A process role. The agent who wrote the review protocol at the review meeting, and updated the Review Report after the meeting.

4. Time: When the review meeting took place.

89
5. **Place:** Where the review meeting took place.

6. **Documents:** A list of reviewed documents, each list element contains the following information:
   
   (a) **Number:** A process internal sequence number.
   
   (b) **Item revision:** The name of the item revision being reviewed.
   
   (c) **Author:** The agent who has written the item revision, this agent is normally also responsible for possible item updates, decided in the review meeting.
   
   (d) **Controller:** The agent who is responsible for controlling the item updates, if any. Both updates and controls are performed after the review meeting.
   
   (e) **Status:** Normally the status decided in the review meeting (OK, FIX, PARTIAL, or NOK (Not OK)). A document with the FIX status may receive the OK status after an update and a control. If the document receives the PARTIAL or NOK status it must go through another DR process after updates.

7. **Scope:** The default scope is the complete documents, but this can be reduced by naming specific document sections which are to be reviewed.

8. **Source documents:** The documents are reviewed against a set of source documents, i.e., they should conformly implement functionality described in these documents.

9. **Informative documents:** Other documents which are needed to understand the reviewed documents, but the reviewed documents are not reviewed against these documents.

10. **Actual effort:** The actual time used on the review process, further divided into: Preparations before the meeting, the meeting, writing the review protocol, and controlling document updates.

11. **Document annotations:**
   
   (a) **Annotation number:** A sequence number local to the process instance.
   
   (b) **Place:** Identification of document, page, and paragraph which needed to be updated.
   
   (c) **Responsible:** The agent which was responsible for the update.
   
   (d) **Annotation:** A short description of the update.

During the process of identifying and updating DOC’s the RR is kept in its initial lifecycle state (UNDER-WORK). It is set to DRAFT by the meeting manager when every FIX document has been updated and controlled. Finally the RR is APPROVED or REJECTED by the project manager, but note that the DOC’s are not approved in the document review process.

### 4.6.2 Roles

The following roles are referenced within the document review activity model:
1. **Author.** The agent that initiates a review process, can be any project member.

2. **Meeting manager.** Responsible for a specific review process in the project.

3. **Secretary.** Responsible for writing the review protocol in a specific review processes in the project.

4. **Reviewers.** Responsible for reading and commenting on documents in a specific review process in the project.

5. **Controller.** Responsible for controlling minor updates after an author has corrected them according to the review protocol (in a specific review process in the project).

All roles in the DR process are relative to a specific process instance. i.e., it has no meaning to talk about the secretary outside the context of a given DR process. Both authors and controllers are relative both to a process instance and a document instance.

### 4.6.3 Activity model

The review process is initiated by an author that wants to have one or more of his documents 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.

The following describes each of the process steps in Figure 4.22:

1. **Create RR.** The author creates a review report (RR). He fills in the name of the review manager, review secretary, and reviewers, writes the scope of the review, and writes time and place for the review meeting. He ensures that every document is set to DRAFT, RR remains in the UNDER_WORK state. 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 (RR). Note that the result for individual documents is documented in the RR, although it is a property of the individual documents. The RR item is updated in PCMS, but the document items are not updated. The updated RR is printed. The RR status is unchanged (UNDER_WORK).

3. **Change document status.** The meeting manager changes the status of documents with review result different from OK. The status is changed from DRAFT to REJECTED. The meeting manager distributes documents according to their review result (OK, NOK, PARTIAL, or FIX).
   - **OK:** These documents need no more attention.
   - **FIX:** These documents need to be corrected, but do not need to be reviewed by an additional review process.
   - **PARTIAL:** Some documented parts of these documents need to be corrected or rewritten, and the updated document must be reviewed by another review process.
   - **NOK:** These documents should be totally rewritten.
4. **Correct and control.** Decomposed. Valid for documents with review result equal to FIX. The process produces new revisions of documents with status set to DRAFT. It may also produce comments on the RR paper copy (the RR item is not updated).

5. **Change RR status.** The meeting manager collects updated documents from possibly different authors. When all OK and FIX documents have been received the meeting manager updates the RR and sets its PCMS status to DRAFT. The other documents (NOK and PARTIAL) needs a new review and are irrelevant to this process.

6. **Approve RR.** The RR is set to APPROVED by the project manager (it is hardly ever rejected). This is the exit criteria for the overall review process. Note that documents are not approved by the review process, this is done by the project manager in some approval process (not explicitly modeled).
**Review meeting:**

1. **Prepare meeting** 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. The activity only manipulates paper copies, not PCMS items.

2. **Meeting** 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 RR). Each document is given a review result, being either OK, NOK, FIX, or PARTIAL. The activity only manipulates paper copies, not PCMS items.

3. **Update RR** The secretary updates the RR (PCMS item) according to the review protocol. The updated RR is printed and handed to the review manager.
The “Correct and control” activity is decomposed in Figure 4.24. There will be one such process for each author with one or more documents that should be corrected (result FIX).

![Figure 4.24: Correct and Control](image)

Correct and control:

1. **Correct document.** The author extracts his documents from PCMS and updates them according to the RR document. New revisions of the documents are created automatically with status UNDER-WORK. The review meeting decided whether the document should be controlled or not, documented in the RR. If control is needed he prints the documents and sends them to the allocated controller, if not, he goes directly to the finish document activity.

2. **Control document.** The controller checks the updated documents according to the RR. She gives the final comments directly on the paper copies and sends them back to the author for final update.

3. **Finish document.** The author makes his final updates and returns the documents to PCMS, he then sets the status of these documents to DRAFT.
4.7 Change management

The *Change management* process can be initiated by the project manager upon changing customer requirements or other external events, provided that these events have consequences for documents with PCMS status APPROVED. The process can also be initiated by a project participant upon discovery of one or more “errors” in one or more approved documents, caused by flaws in previous activities. Regardless of reason, if a PCMS item must be updated and it has PCMS lifecycle state equal to APPROVED, one must execute a change management process.

The following describes the *Change management* process in detail. First, the involved documents are described, namely the *Change Report* (CR) and the *affected documents* (DOCs). Second, the change management roles are described. Finally, the developed models are depicted and explained.

4.7.1 Documents

The change management process involves two types of documents, the Change Report (CR) and the documents to be changed (DOCs). For each change management process instance there is exactly one CR, but possibly many DOC’s. Note that DOC denotes any PCMS item, of any PCMS item-type. The CR keeps process information and will be updated as the process moves forward, e.g. every involved DOC is referenced in the CR. A process instance is finished when the new item revision of every affected document has PCMS status APPROVED.

Within the Company, each CR is implemented as an ordinary PCMS item, not as a PCMS Change document (explained in Chapter 6). For each project, all CR’s are stored under a common PCMS part, and are named with “CR” plus a sequence number local to the project (e.g. CR-32). In the studied project there were 102 such items. The items has PCMS type C3 (QA-archive) and the DOC-1 lifecycle.

After executing an instance of the change management process, the corresponding Change Report contains the following information:

1. *Problem or change description*: A textual description of the problem or proposed change. This can be divided into sub-problems, in this case the description is made as an enumerated list of problems or proposed changes.

2. *Solution*: A textual description of the solution. If the problem is described as a list of sub-problems the solutions are enumerated accordingly.

3. *Change responsible*: The agent who was responsible for this instance of the change management process.

4. *Change status*: When the process ends this field contains the DONE value. During the process, the values might be UNDER WORK, SOLUTION APPROVED, SUSPENDED, DONE.
5. **Affected documents**: A list of documents that have been updated by this instance of the change management process. Each list element contains:

   (a) **Item revision**: The name of the item revision that is updated (the old item revision).

   (b) **Responsible**: The agent who was responsible for changing this item revision, producing a new item revision.

6. **Revealing activity**: The name of the activity in which the problem was discovered.

7. **Estimated effort**: The estimated number of hours to fix the problem.

8. **Actual effort**: The actual number of hours used to fix the problem.

9. **Introducing activity**: If more than one problem is reported there is an associated list of problems, each list element contains:

   (a) **Problem number**: A problem sequence number according to the problem description.

   (b) **Introducing activity**: The name of the activity which introduced the problem.

   (c) **Category**: A classification of changes according to their cause. The alternatives are: external change (customer), internal change (simplifications, structuring), error, interpretations (if the specification was ambiguous).

During the process of identifying and updating DOC’s the CR is kept in its initial lifecycle state (UNDER-WORK). It is set to DRAFT by the change responsible when every document has been updated and controlled. Finally, it is APPROVED or REJECTED by the change manager.

### 4.7.2 Roles

The following roles are referenced within the change management activity model:

1. **Change manager**: Responsible for all change management processes in a given project. Often identical to the project manager.

2. **Change reporter**: The agent that initiates a change management process, it can be any project member.

3. **Change responsible**: Responsible for a specific change management process in the project.

4. **Editor**: Responsible for updating one or more documents within a specific change management process. There might be many editors within a single change management process.

A Change manager is responsible for all Change Management processes (type) while the Change responsible is responsible for a single process (instance). The Editor is a role that is both relative to a process instance and a document instance.
4.7.3 Activity model

The overall Change management process is initiated by a project participant finding one or more “errors” in one or more approved documents. It is not necessarily a flaw in previous activities, it may also be caused by changing requirements leading to rework.

In the following we describe each of the process steps in Figure 4.25:

1. **Initiate change.** Decomposed. The Change Report is created and a change responsible allocated. Affected documents might be identified and referenced by the CR. A paper copy of the CR is sent to the change responsible.

2. **Find solution.** The change responsible is responsible for finding a solution to the problem. He might approve the solution suggested by the change reporter, he may find a solution himself, or he might call for a meeting where the problem is discussed. When a satisfactory solution is found he extracts the CR from PCMS and describes the solution. He must ensure that all documents affected are listed in the CR with correct revision numbers, and that every document is allocated a document editor. The CR status field is set to SOLUTION-APPROVED. Sometimes it is necessary to suspend the changes until some condition is satisfied, this is a decision of the change responsible.

3. **Suspend changes.** Decomposed. If the change responsible decides to suspend the CR he is required to provide a criterion for its reactivation. This criteria is monitored by the change manager responsible for the reactivation. When reactivated, the CR is checked and possibly changed, the process then continues as is if it had not been suspended. A paper copy of the CR is sent to involved editors.

4. **Make changes.** Decomposed. The different editors receive a paper copy of the CR. On the basis of information found in the CR, they extract referenced documents from PCMS. The editors make the necessary changes until the change responsible is satisfied. The CR might possibly be changed by the change responsible. Then the editors return their documents to PCMS and set document statuses to DRAFT.

5. **Finish CR.** When every editor has changed every document, and every document has obtained PCMS status DRAFT, the change responsible changes the CR status field from “SOLUTION-APPROVED” to “DONE”. He also changes the PCMS status from UNDER_WORK to DRAFT.

6. **Approve changes.** Decomposed. When the CR has status “DONE”, the change manager is to approve every document (PCMS status APPROVED), and then finally to approve the CR (PCMS status APPROVED).
Figure 4.25: Change management

- Find solution
- Change reporter
- Change manager
- Change responsible
- Editors
- Finish CR
- Approve changes
- Make changes
- Change manager
- Change responsible
- Approved documents from PCMS repository
- Suspend
- Initiate change
- not suspended
- change manager
- change responsible
- change manager
- change responsible
- change manager
Initiate change:

1. **Create CR.** The change management process is initiated by a change reporter. The change reporter creates a Change Report, and as a minimum he fills in a problem description. He might also provide other types of information that he has knowledge about at this point in time, e.g., affected documents. The CR status field is set to UNDER-WORK, PCMS status is UNDER-WORK. A paper copy of the CR is delivered to the change manager.

2. **Appoint Change Responsible.** The change manager appoints a change responsible. The change responsible could have been suggested by the change reporter, in this case the suggestion is either accepted or a new one appointed. The CR is annotated with the decision and a copy sent to the change responsible.
Suspend changes:

1. **Suspend CR.** If the change responsible decides to suspend the CR he is required to update the CR status field from “SOLUTION-APPROVED” to “SUSPENDED until <condition>”. He then changes the PCMS status from UNDER-WORK to DRAFT, while he is also required to write SUSPENDED until <condition> in the “Comment” field of the PCMS Action item command.

2. **Reactivate CR.** The change manager is required to monitor suspended CR documents and to reactivate the corresponding change process. When the change manager reactivates a change process after it has been suspended, it is again handed to the change responsible.

3. **Control CR.** The change responsible is required to check the CR to ensure that the provided information is still valid, if not he will perform the same actions as in “Check CR”. When satisfied, he will change the CR status field from “SUSPENDED until <condition>” back to “SOLUTION-APPROVED”. He will then change the PCMS status of the CR from DRAFT to UNDER-WORK, utilizing a PCMS feature that does not create a new revision. The CR and affected documents are distributed to the respective editors.
Figure 4.28: Make changes

Make changes:

1. **Change documents.** An appointed editor extracts the documents he is responsible for from PCMS, new revisions are automatically created with PCMS status UNDER-WORK. He then performs the necessary updates. He is not finished before the changes have been approved by the change responsible. He communicates with the change responsible by means of paper copies of the updated documents.

2. **Control documents.** The change responsible reads and comments on documents from the various editors upon request. There might be several iterations before the change responsible is satisfied. When the change responsible satisfied, the editor is required to change the PCMS status of his documents.

3. **Change document status.** When the change responsible is satisfied, the editor is required to change the PCMS status of his documents from UNDER-WORK to DRAFT. He is also required to write a reference to the CR in the “Comment” field of the PCMS Action item command.

The correct APEL interpretation of Figure 4.28 is that the *Change documents* finishes, before *Control documents* is allowed to start. If the controlled documents are NOK when the *Control documents* finishes, the *Change documents* activity restarts. However, seen from the Company point of view, the *Change documents* activity is active until the *control documents* activity has accepted the updated documents.

Thus, if APEL semantics were to be strictly applied, the *Change documents* and *Control documents* activities would have to be modeled in parallel, synchronized by events.
Approve changes:

1. **Approve documents.** When the CR is “DONE”, the change manager approves the documents by setting the PCMS status of the involved documents to APPROVED (from DRAFT). He may also REJECT the changes, which requires a new change process to be performed.

2. **Approve CR.** When every document is approved the change manager will set the CR to APPROVED. If not approved he will set the CR to REJECTED.

### 4.8 Summary

This Chapter started by explaining the applied process modeling language in terms of its syntax and semantics. The language is based on the APEL language, but adapted to suit specific modeling needs. The major differences with respect to the APEL semantics is as follows:

- The control flow does not imply strict chaining, i.e., activities related by a control flow can be active at the same time.
- Data input and output are interpreted in terms of PCMS operations.
- Paper copies of documents are explicitly modeled.

The adapted syntax and semantics reflect modeling experiences. I preferred constructs that were understandable to the organization, and which resulted in simple diagrams. In this way, the resulting process modeling language becomes natural to the process owners.

This Chapter explains the selected modeling constructs to be able to read the models. In the next Chapter I report experiences with the selected set of concepts, and furthermore, experiences from the modeling process.
Chapter 5

Modeling experiences

The previous Chapter described the APEL modeling language and expressed the Engineering Handbook processes by means of this language. Our APEL models correspond to Engineering Handbook process descriptions as I mainly rephrased the models with APEL syntax and semantics, and not extended the modeling scope. This chapter describes problems encountered and lessons learned during the process modeling activity.

I have divided the experiences according to three APEL modeling aspects, namely the activity, product, and human resource aspects of the software process. Section 5.1 discusses activities, activity chaining, and activity decomposition. Section 5.2 discusses differences between the product models of the Engineering Handbook and PCMS, and furthermore, the influence of configuration management practices with respect to process modeling. Section 5.3 discusses the modeling of human resources in terms of the relationship between activities, roles, teams, and agents. Section 5.4 describes the overall experiences with the APEL modeling environment. Section 5.5 discusses the need for a distinction between type and instance level models.

5.1 Activities

This section describes experiences related to the APEL activity model and the application of these concepts to Company processes. The section is organized as follows: Section 5.1.1 discusses the use of the activity concept in terms of inputs and outputs, enter and exit criteria, and multiple instances. Section 5.1.2 discusses aspects of activity chaining, data and control, sequence, concurrency, choice, and feedback. Section 5.1.3 discusses activity decomposition criteria and the concept of detailed activities.

5.1.1 The activity concept

This section describes experiences with the activity concept as it was used in the Engineering Handbook and in the developed APEL models described in the previous chapter. By the activity concept we here mean a single activity, its inputs and outputs, enter and exit criteria,
and multiplicity.

**Activity inputs and outputs**

Activities were described as a set of input and output document types, together with a free text activity description. The most typical situation was that a set of input documents should be used as the “information resource”, from which the “human resource” was to produce another set of documents that were the “goal” of that activity. If further detailing was needed, it was described in terms of conditions associated to document properties (e.g. approved, rejected).

**Output documents:** We found a close relationship between the naming of output documents and activities, e.g. the activity “Make BID” was to produce one or more documents of type BID (System design, Figure 4.9, page 75). In this way the definition of the output easily becomes the definition of the activity, and an accurate definition of the output becomes important. The outputs were defined as document templates, but in practice documents were included in other documents and were not traceable as PCMS items (see section 5.2.1).

**Input documents:** We observed that defined outputs of an activity often were more in correspondence with reality than its inputs. The reason for this is that it is much more difficult to decide what to include in the input list, than in the output list. E.g. should only documents that are extracted from the repository be included? We found three input categories:

1. A *triggering document* is a document whose existence or status will cause an activity to be enabled (allowed to start).

2. A *background document* is a document that is generally useful for most tasks, e.g. the Customer Requirement Specification (CRS). Such documents are normally available when needed, thus not regarded as triggering (not on the critical path). Some documents are used as a triggering document in one activity (CRS for System Design) but as background information in later phases. Some documents are only used as background information, e.g. the Engineering Handbook (EH). In some cases the use of background information is indicated in the process model, interpreted as “typically used in this activity”.

3. A *middle document* is not needed to start a given activity, but must be present before it is allowed to finish. I.e., we may start the activity without it, but at some point we will need the document in order to proceed. This is generally true when there are more than one input document which are not background documents.

When documents are manipulated electronically they will be extracted from (input) and returned to (output) the repository, thus producing historical information. This is not the case when documents exist as paper copies.

**Paper copies:** Input documents that are considered as specifications or background information normally exist as printed paper copies. These documents are not extracted from the repository, and the repository will not contain any information about their use. In some cases also the output documents only exist as paper copies. There are two major cases: First,
when documents are read and commented on by handwriting, e.g., reviewed and controlled
documents. Second, when authorizations or decisions need signatures or filled in forms, e.g.,
when allocating the change responsible in change management. Thus, to register input or
output documents of activities it is not sufficient to rely on extract and return of repository
items. To capture input and output information the process agents must also actively register
paper documents. However, in the case when every document exists in the repository, it is
sufficient to register the corresponding repository item.

Enter and exit criteria

Since every document in the Company is put under configuration management control (PCMS)
it is natural to use the status of a PCMS item as enter and exit criteria. Generally, we found
that any enter and exit criteria could be associated with either document properties, human
resources, or time. However, only the document properties were important to a type level
description. We also found that any enter and exit criteria could be overruled by humans,
typically by the project manager.

- Enter criteria: The project manager starts a task when he feels confident that the task
can be done, and when there are agents available, not working on tasks with higher
priority. The task can be done when certain conditions are satisfied, a typical criterion
is the availability of certain documents with the appropriate status. When an agent is
told to start working on a task, he is interested in the relevant background information,
i.e., certain documents of a certain quality. Thus, the definition of a task should act as a
guideline for the project manager, and later as a contract between the project manager
and the responsible agent. In both cases the important issue is the existence and status
of input documents and whether these documents conform to the activity description.

- Exit criteria: A task should finish when the goal is achieved, i.e., when the required
output exists. As for inputs this means that a certain set of documents exist in a given
state. A task can also be forced to terminate before the goal is achieved, e.g., by the
project manager. In this case the exit criterion will be false, but there might exist useful
results.

To the extent it was useful to model enter and exit conditions, these were modeled as logical
expressions over PCMS life-cycle states or attribute values associated to document types.
This was applied within the Change management and Document review processes. In the
System development processes this was only formalized upon phase endings, where every
phase document must have PCMS status APPROVED. At a decomposed level the general
rule is that documents receive PCMS status DRAFT after a document review process. How-
ever, a DRAFT document can be updated at any time by the producing activity, which is
not regarded as terminated before the documents are APPROVED. At this point, Change
management processes must be executed to change these documents.

To conclude, on the type level it is sufficient to model the enter and exit criteria with respect
to enter and exit criteria on document types. Other information, such as named items, named
human resources, and calendar time, is information which does not fit into type level models.
We advocate explicit instance level modeling to capture such properties, this because these

106
properties are specific to a particular project setting, and are subject to frequent change. By instance level modeling we mean modeling similar to what is applied in project management software, e.g., in the form of activity networks [89, 88].

Activity multiplicity

APEL differentiates between activities which have one instance, and those which have many instances during process execution. This is also used in the developed models. However, I experienced that the distinction is artificial because almost any activity could have several instances. Thus, the modeled multiplicity must be understood as “these activities usually have more than one instance”, while the single multiplicity as “these activities usually have only one instance”. Activity multiplicity is further discussed in Section 5.5, page 130.

5.1.2 Activity chaining

This section discuss chaining dependencies among activities, decomposition is discussed in section 5.1.2. First, the experiences with APEL’s distinction between data and control is described. Then the use of chaining concepts such as sequence, parallelism, choice, and feedback are described.

Data and control

APEL separates strongly between data and control, the distinction can be described as follows:

1. Control. A control flow describes the precedence relationship between activities. I.e., which activity must terminate before another activity may start.

2. Data. A data flow describes the producer/consumer relationship between activities. I.e., which activity produced the document that is consumed by another activity.

Due to APEL semantics we applied the distinction during our modeling. However, our experience is rather negative – it did not contribute to process understanding – it just complicated our diagrams. The separation between data and control flows caused the diagrams to be more difficult to read, difficult to update, and caused unintended levels of decomposition. They were difficult to read and update because of the large number of graphical links among activities.

The unintended levels of decomposition were caused by a need to include the diagrams in EH documents. APEL diagrams used more space than the original diagrams, thus, we needed to redefine the original decomposition hierarchy to make fewer activities at each level. One example of an “artificial” decomposition is the software development activity depicted in Figure 4.13, page 80.

Our experiences with graphical activity modeling are that models should be simple, understandable, easy to create, and easy to change. Specifically, we found that more than one relationship between two activities invalidates these properties. We then asked the following
question: If this is generally true, what should this relationship express? We found the answer to be document transportation, illustrating that a producer activity is responsible for the transportation of documents to a consumer activity. This is further discussed in Section 5.5.1, page 132.

Sequential activities

In the Engineering Handbook, sequencing of activities was always modeled as document dependencies. Activities were producers and consumers, and links between activities denoted the distribution of a document from a producer to one or more consumers. The communication channel was either the repository or – in case of hard-copies – personal distribution.

In the project management literature, activity dependencies are often categorized into start–start, start–finish, finish–start, and finish–finish, the meaning being that the former must start before the second can start, etc. A strict sequence is obtained by applying the finish–start dependency. In the studied case we identified the following two kinds of dependencies:

- The finish–finish dependency was quite common, e.g., development phases were related in this way. A finish–finish dependency may cause concurrent, sequential activities, this situation occurs when there exists a producer–consumer relationship between two activities and the consumer starts before the producer finishes.

- The finish–start dependency was applied only in the Change Management and Document Review processes. We here mean always applied, in other processes only specific instances were related in this way.

The control aspect of a document is the set of events caused by document creation and changes in the revision history. Ideally, documents should be APPROVED before they were used by a succeeding activity, in practice they were normally DRAFT. Generally, we can specify exact start-up criteria, but we cannot expect these to be exactly followed by the project manager. The final decision is made by a human being, having the following implications:

1. A task does not necessarily start when the prescribed input is available. The enter criterion is true, but the task is not active.

2. A task may start before the prescribed input is available. The enter criterion is false, but the task is active.

3. A task may end before the prescribed output is produced. The exit criterion is false, but the task is finished.

4. A task may be active even after the prescribed output is produced. The exit criterion is true, but the task is active.

In general, using process enactment to enforce strict chaining would not be accepted by project personnel. This because of the dominating finish–finish criteria, and uncertainties related to activity instantiation and binding. However, there were two notable exceptions, namely the Change management and the Document review processes. These two processes were already applying strict chaining and were well defined with respect to instantiation and binding.
Parallel activities

Parallel activities were quite common in the Engineering Handbook. Copies of the same documents were often distributed from a preceding activity to different threads of control and later collected in a synchronizing activity. However, parallel activities are not necessarily independent.

![Diagram](image)

Figure 5.1: Parallel activities – Functional analysis

**Parallel, dependent activities.** Figure 5.1 shows the activity model for Functional analysis. Activity 1 produces an SRS type of document which is consumed by the parallel activities 2, 3, and 4. Activity 2 and 3 update this document, while activity 4 produces another document of type SID. In such situations, activity inputs and outputs often give hints about activity dependencies:

1. **Same input, same output.** This was used as a strong indication that more than one agent should be used (or was normally used), that they produce different parts of a single document, and that they need to communicate closely. E.g., activity 2 and 4 in Figure 5.1.

2. **Same input, different output.** When two or more activities had different outputs they seem to be less dependent. However, we still see examples that communication was an important issue. E.g., the two documents made by 2 and 3 in Figure 5.1 were dependent upon each other.

3. **Different input, same output.** In these cases the “same output” was only at the type level, at the instance level there was no overlap. E.g., the MRS and BO inputs, and BO outputs of the activities numbered as 2 and 3 in Figure 4.15, page 83. Here the actual output documents are of the same type (BO) but the two activities are concerned about different document instances.

4. **Different input, different output.** In this case one could assume that there was little need for inter-activity communication. E.g., the activities numbered as 7 and 8 in Figure 4.14, page 82.
In the Engineering Handbook I found a tendency to model activities that were usually performed in sequence as sequential activities, even if they were not inherently sequential regarding inputs and outputs. This represented a problem because I had to choose between the model that is “natural” to the process owner, and what is “correct” with respect to document dependencies. This was a problem when modeling the System design phase depicted in Figure 4.14 (page 82, activity 1 and 2). However, I decided to model the activities according to document dependencies, modeling activity 1 and 2 as parallel activities.

Multi-instance parallelism. Multi-instance parallelism occurs when there is an unknown number of tasks for each activity. E.g., if there are two sequential activities A and B, but there can be many A tasks and B tasks. The question is which output from which A task is bound to which input of which B task. This is generally a run-time decision that must be made by some human, e.g., in a separate binding task. Multi-instance parallelism is further discussed in Section 5.5, page 130.

A guideline for choosing parallel activities is that the activities have the main responsibility for different document types. A guideline for multi-instance parallelism is that the activity instances have the responsibility for different document instances.

Alternative activities

A choice is a decision point where not all of the following process threads are necessarily activated. I found little use of choices, however, some were found, e.g., in Implement and test modules, Integration, Change management, and Document review. Characteristically, these processes were well understood and modeled in the greatest detail. In the modeled activities, choices were always documented as values associated to one or more documents.

Example: In the Implement and test modules activity depicted in Figure 4.19 (page 86) there is a choice before activity 1, and another after activity 4. The choice after activity 4 depends upon the PCMS lifecycle state of documents of type MTP after module test. If all MTP documents are APPROVED, the overall activity should terminate. If one or more MTP documents are REJECTED the activity is not allowed to terminate. In the latter case one must start a Change management process for the documents which caused the test to fail. The type of candidate documents are SB and ADTSC. The Change management process will identify the affected documents and distribute the responsibility for document updates among project participants. The allocated agents will then perform all the four activities of Figure 4.19, starting with the control OR symbol before activity 1. □

Example: In the Integration activity depicted in Figure 4.20 (page 87) the situation is identical to the Implement and test modules activity, but in this case the criterion for the choice after Verify ITP is on documents of type ITP. □

Example: In the Document review activity depicted in Figure 4.22 (page 93) the execution of Correct and control depends upon the value of the review status attribute of the reviewed documents. If all DOCs are OK the Correct and control activity is not executed. If one or more DOCs are FIX, Correct and control is executed, but then only on FIX documents. At the decomposed level, depicted in Figure 4.24 (page 95) the execution of Control document depends on the value of the controller attribute of the reviewed documents. The activity is

110
only performed on documents with an allocated controller. □

**Example:** In the *Change management* activity depicted in Figure 4.25 (page 99) the execution of *Suspend changes* depends on a decision made by the *change responsible*. The decision is documented in the *change status* attribute in the Change Report (SUSPENDED), and in the CR’s PCMS action description when the CR is actioned from UNDER-WORK to DRAFT. When the process is reactivated, the *change status* is updated to SOLUTION-APPROVED. □

**Example:** At a decomposed level in the *Change management* activity there is a choice in the *Make changes* activity, depicted in Figure 4.28 (page 102). In this case the controller reads a print-out of the updated document and writes comments on the paper. □

**Repetitive activities**

Repetition occurs when we have a loop in the graph describing activity dependencies. Here we found the most surprising observation; the Engineering Handbook did not have loops in the activity graphs. Rather than using loops, the EH described two processes which handled two different kinds of repetition, namely *feedback* and *rework*.

1. **Feedback – Document review.** A document produced by a task is required to go through at least one document review process. The initiating task is regarded as active until the review is finished. If major updates are needed, the activity continues and later resubmits the documents to a new document review process. If all documents associated to an activity are reviewed and found to be OK, they will given the PCMS lifecycle state DRAFT. At this point the activity is considered to be finished – in the sense that no more work is expected – but the activity is not formally finished before all documents are given the PCMS lifecycle state APPROVED.

2. **Rework – Change management.** If an error is found in a document that has been APPROVED it can only be changed by means of a change management process, producing a new APPROVED revision. The execution of a change management process may require that other processes execute, but then new tasks within the context of the change management process.

An observation is that feedback is used to model “small loops”, limited in time and effort, while larger rework is better described as creating a new task to fix the problem.

**Example:** When the *Implement and test modules* activity depicted in Figure 4.19 (page 86) fails, a *Change management* activity is started. However, the execution of the *Change management* process involves the execution of a new *Implement and test modules* process, but then performed within the context of the *Change management* process. □

The case study observations with respect to *feedback* and *rework* are interesting with respect to the interpretation of loops in activity graphs. In the following we argue that loops should not be interpreted as the iteration concept found in programming languages, but rather as communication between active tasks (feedback) or the creation of new tasks (rework).
In a computer program *iteration* means that previously executed code is executed again, usually on data that have changed since the last execution. If we compare computer programs with process descriptions, we must see the agents as the processing element, and that the agent is to perform exactly the same operations on possibly different data. I find this not to be a useful approximation of human behavior. I find it more useful to think of human work as starting a task, doing the work, and then at some point finishing the task. When a task finishes it will never be executed again. I find the APEL *iteration* to be counterintuitive and that it should not be used as a process modeling concept.

However, one should distinguish between iteration and *feedback*. A task should only receive feedback while it is active, a task that finishes will never be executed again, and cannot receive feedback. When a task has finished and there is a need to fix a problem in the task's output, a new task should be created with the old output as input. This new task represents *rework* because it would not have been created if the old output could have been used.

### 5.1.3 Activity decomposition

This section discusses activity decomposition and its relationship to activity detailing and correctness. First, I discuss properties of the leaf-level activities, as found in the lowest level of decomposition of our APEL models. This corresponds to the lowest level of decomposition found in the Engineering Handbook as I only rephrased the models with APEL syntax and semantics and did not extend the modeling scope. Second, I distinguish between leaf-level activities and detailed activities by introducing the concept of an operational description, and furthermore, discuss properties of a detailed activity. Third, I make a distinction between operational descriptions and correct descriptions, seen from the perspective of the actual development process. Finally, I discuss *why* some activities are modeled in more detail than others.

#### Properties of leaf-level activities

By investigating the activity hierarchies one finds the activity trees not to be balanced, the minimum number of levels is 2 and the maximum is 6. Studying the leaf-level activities in the developed APEL models one finds the following *ranked* list of common characteristics:

1. *Number of involved agents.* A leaf-level activity involves a single agent. Where activities are instantiated into parallel tasks, the tasks can be distributed among several agents, but not more than one person on each task.

2. *Number of produced products.* A leaf-level activity has a small number of output document types (one or two). Where activities are instantiated into parallel tasks, sub-sets of document instances are distributed on these tasks.

3. *Expected duration.* A leaf-level activity has a short duration (days or hours).

The most prominent Company criterion for deciding when to stop the activity decomposition is the "single person" criterion.
Example: The Project management activity produces a wide range of documents, but is done by a single person, thus no decomposition. □

Example: The Analysis activity produces less documents but involves more people, thus a decomposition exists. The Analysis decomposition includes Implementation analysis which produces one document (not decomposed), and the Functional analysis which produces two documents (decomposed). □

Properties of detailed activities

Many decomposition levels should not be confused with high-level of detail. I find it useful to think of a detailed activity as an activity with an operational description. By operational I mean to what extent a process “freshman” can perform the process according to organizational practices, that is, if he had to perform the process solely on the basis of its description. In this perspective I find the three criteria – identified above – of equal importance. The single agent criterion is important because then the synchronization with other agents is resolved in the activity model the activity is part of. The single document criterion is important because then the task is likely to be coherent, i.e., the agent does not need to work on many different things within the task. The short duration criterion is important because then the task is more achievable, and not easily influenced by project priorities, changing goals, and resource priorities.

Example: The “Project management” activity is not detailed. It satisfies the single person criterion, but violates both the single document criterion and the short duration criterion. □

Example: The “Make SIDs” activity depicted in Figure 5.1 (page 109) is detailed. This activity satisfies all the three criteria. □

Note that an operational process description is not necessarily an enactable process description, neither is an enactable process necessarily operational. E.g., we may perfectly well have an operational description of a project meeting that cannot be implemented as a process program. Furthermore, we may perfectly well implement a single activity called “project” as a process program without having an operational process, e.g., a program which registers the name of the project and terminates.

Activity correctness

Correspondence with reality is an important property of high-quality process models, and we may define correctness as the lack of difference between the process description and the actual process. Correctness is a property related to, but not identical to operationability. Their differences depend upon how established a process is within the organization. A process is well established if an activity is performed in the same way for all activity instances, regardless of whether it is described or not. If a process is well established, an operational description will be a correct description (by the definition given above).

However, if a process is not well established within a company, the concept of operationability loses some of its meaning. In this case an operational description can only capture one possible way to perform the work, since there is no coherent process. Thus, with respect to the actual
process, an operational description is not necessary correct.

Furthermore, a correct process description is not necessarily operational. This can be illustrated by taking it into its extreme, e.g., we might say that an undescribed activity is close to reality because any action will be conformant to the description. In this case we can say that the description is correct, but not operational. (However, this is not necessarily bad because the process modeler should not “invent” processes that do not exist).

Example: The Project management activity is correct but not operational. It is not operational because it is not described in the EH (other than by its name and its products).

□

Detailing criteria

I have established some level of detail criteria for software engineering processes, and will now comment on why different parts of the process are described at different levels. E.g., are some processes more important, better understood, or more difficult than others? The most viable explanation is that the level of detail depends upon the perceived Company needs. Such needs can be classified as:

1. Knowledge needs: These are covered by the traditional Handbook style of descriptions, characteristically containing a free text description of activities, procedures, standards, guidelines, experiences, hints, etc. Usually it is defended as being important to new personnel, implicitly: “most of us do not use it”. Parts of the Engineering Handbook have this profile, e.g., the informal style of what to do, guidelines for how to do it, standards, and hints.

2. Quality assurance needs: These are covered by the “Quality manual” style of descriptions. Such handbooks are more concerned about strict rules for how certain things are done, presenting formal procedures that are expected to be followed. A properly defined quality system has built-in procedures to check adherence to standards, and to take actions upon discrepancies. This style is most prominent in the Change management and Document review parts of the Engineering Handbook, but also in integration, detailed design and implementation.

3. Improvement needs: These are covered by the “GQM-plan” style of description. The focus is on defining goals and metrics, and the operational aspects such as data gathering, validation, and analysis. Although it depends on the measurements, it usually requires a well defined and stable software process. With respect to the Engineering Handbook, measurements were described in the Change management and Review processes. Effort was also measured according to EH activities. Additionally, measurements were described in the GQM-plan document, and the major process issue was to have well defined and stable development phases.

Although all aspects were present in the Engineering Handbook, its major focus was on knowledge needs (except for the change and review processes). The major reason for this is that quality assurance and improvement needs would have been satisfied by a description of
phases. To be more precise, lower level activities were not checked for conformance, neither were they subjected to any other measurement activity.

The organization itself stated that the major rationale for the Engineering Handbook was process improvement needs, but this seems to be more a goal than reality. On the background of our modeling activity I formulated some theories about differences in level of detail. The following “effects” are partially overlapping:

- **The understanding effect.** The level of detail depends upon how well the various processes are understood. Processes that repeat themselves within a project or between projects are easier to generalize into a process model.

- **The coordination effect.** The level of detail depends upon task size, document volume, and the number of agents involved. Processes with many small tasks, involving many documents and people are likely to have a more detailed description.

- **The certification effect.** The level of detail was the one considered to be sufficient to receive a quality certificate, e.g., ISO-9000.

- **The management effect.** The level of detail depends on the status of people that typically perform that part of the process. Essentially, one does not model the processes of “important” personnel.

The first two could be seen as rational behavior, while the latter two are more unconscious or irrational effects.

### 5.2 Products

This section discusses product modeling on the background of the PCMS repository information and experiences with APEL product modeling. Product modeling is discussed from the viewpoint of the studied organization, and PCMS is therefore used as a reference throughout this section.

This section is organized as follows: Section 5.2.1 discusses the difference between EH product concepts and their manifestation in the PCMS repository. Section 5.2.2 discusses the usefulness of product modeling in a situation where the product model cannot be represented in the repository. Section 5.2.3 discusses the role of product lifecycles and revisions. Section 5.2.4 discusses criteria for choosing product abstractions.

#### 5.2.1 Concepts and manifestation

Here I discuss the difference between organizational concepts and their manifestation in the PCMS repository. This discussion is limited to organizational concepts which are documented in the Engineering Handbook. In the Engineering Handbook, the organizational concepts were presented in a structured way by means of a list of abbreviations. This list includes the product types used in the activity models. For each product type one should expect to find corresponding items in the repository, hereafter denoted as the manifestation of the concept.
Direct and indirect manifestation

A direct manifestation occurs when organizational concepts are modeled as repository concepts and their instances are handled by the repository. An indirect manifestation occurs when the organization maintains a description of how the concept materializes in the repository, and how instances can be deduced from repository information.

The input and output product types used in the Engineering Handbook had no corresponding PCMS part-type or item-type. Thus, if we view the EH product abbreviations as product types, there are no direct manifestations of EH concepts in the PCMS repository. The consequence of not having direct manifestations is that PCMS reports cannot be made on EH defined product types. Since the EH product abbreviations were coded into item names, there is indirect manifestations. In this case the corresponding items can be found by inspecting the item names in the repository.

Examples: The System Requirement Specification (SRS) is an organizational concept that materializes as a single PCMS item. The Block Interaction Diagram (BID) is an organizational concept that materializes as a set of PCMS items, possibly in different PCMS parts. The Engineering Handbook (EH) is an organizational concept that materializes as a single PCMS part. □

In some cases, there were EH product types which did not have any documented manifestation (direct or indirect). In such situations one do not find any item names which match the EH product type. In these cases one can only find the mapping by inspecting document contents, or by means of explanations given by process agents.

Example: The Block Requirement Specification (BRS) is an organizational concept denoting the requirements on an SDL block. In one product structure the BRS materialized as a PCMS item. In another product structure the BRS materialized as sections in the MRS document, the MRS materializing as a PCMS item. The organizational concept was not changed, only its manifestation. □

The lesson learned is that if the repository is to be used for product assessments, there must exist a mapping between product types used in activity input and outputs, and the repository parts and items. Preferably, the process modeler should only use input and output types defined as PCMS part-types or item-types within the corresponding PCMS control plan.

Manifestation and granularity

The most fine grained PCMS concept is the item-id, identifying a single item revision which can be extracted into a file in the user workspace. The PCMS product-id and part-id are more coarse grained concepts, identifying sets of PCMS items. These cannot be treated as a single entity in the user workspace. E.g., in PCMS it is impossible to extract and return parts, only the individual items within a part. From an activity input/output perspective, process agents work with item revisions. Thus, to obtain an operational process model one should model items and item types.

Within a type-level process model, item revisions must be represented by item-types. An operational type-level model should then have a one-one correspondence between activity
inputs/outputs and PCMS item types. If this is the case, process agents can more easily understand activity input and output types, because they have a PCMS interpretation. This reduces the risk of mistakes and increases the likelihood of useful feedback to the process modeler.

Organizational concepts can sometimes be more fine grained and sometimes more coarse grained than their manifestation as PCMS items. In the Engineering Handbook the granularity of concepts was decided by the process modelling activity, hiding practical considerations by means of abstraction. The granularity of the PCMS manifestation was strongly influenced by the supporting tools, mainly SDT and PCMS. SDT produces files with a given format, according to SDL structures, deciding the granularity of PCMS items. However, in some cases the files produced by SDL were considered at a too low level to be included in the process model, and an abstraction was created (e.g. BTP).

Example: The BTP concept materialized as three types of SDT generated files: BTDMP, BTINC, and BTLOG. This is an example of the BTP concept being more coarse grained than its PCMS manifestation. □

PCMS granularity is also influenced by the document size. If documents become big they may be split into several PCMS items (e.g. EH), if they become small they may be merged into one PCMS item (e.g. BRS).

Example: (Organizational concept more coarse grained). The EH concept materialized as a PCMS part, containing several PCMS items. Seen from the word processor, each of the PCMS items contains an independent document (e.g., a phase document). The items can also be merged in the word processor to create a single EH document. □

Example: (Organizational concept more fine grained). The BRS concept materialized as sections within a MRS document. In this case the BRS document was considered as too small to have its own PCMS item, indicating a need to reduce PCMS overhead. It can also be explained by missing PCMS and/or word processor functionality as the tools are not able to maintain two PCMS items seen as one document. □

The differences between product types in the EH activity model and their manifestation as PCMS items can be explained by a mismatch between PCMS concepts and the concepts needed by the organization. However, PCMS represents state of the art in commercial configuration management systems, and its presence in the organization is a fact in the foreseeable future. Thus, it is useful to exploit the existing PCMS functionality. The differences can also be explained by a lack of strategy within the organization with respect to the use of PCMS concepts. The latter is supported by the fact that PCMS types did not reflect EH definitions, and furthermore, did not serve any other purpose.

To summarize, product abstractions and structuring exists within companies regardless of the corresponding repository support. One can say that concepts are idealized, while their manifestations are subject to practical considerations. Concepts are useful to the organization because they are stable across different implementations, but they create problems with respect to tracking their manifestation as PCMS items. To maximize the benefits of a repository companies should select abstractions that either match the repository, or where a manual mapping can be provided.
5.2.2 Product modeling

The previous section discussed the difference between the product input and output types used in the EH activity models and their manifestation in the PCMS repository. This section discusses the usefulness of the APEL product modeling in a situation where the PCMS repository is a fact. First, the product modeling is discussed in terms of classification, aggregation, and association. Second, the role of product modeling is discussed with respect to the studied case.

Classification

The Engineering Handbook classifies concepts in several dimensions. The EH classification is partly documented in the EH concept abbreviation list, and partly in the textual description of activities.

Example: The Block Interaction Diagram (BID) is classified as an SDT file (SDT), Message Sequence Chart (MCS), and Software Document (SD). The term BID is used to denote activity input and output types in EH activity models. In this context the BID is the result from applying the SDT tool in order to create message sequence charts on the SDL block decomposition level. The term SDT file is used to show that the product is created by means of the SDT case tool, but the case tool also produce files with different content (e.g. SB). The term MCS is used to denote the applied technique, but this technique is also applied to other document types (e.g. SID). The term SD is used to distinguish between hardware, software, and administratie documents. □

PCMS supports single classification by product typing. To obtain multiple classification in PCMS one can use item-type attributes, e.g. one attribute for each possible classification. One can also manually maintain a mapping table in the EH between EH concepts and PCMS item-types, e.g. for each EH concept, list the PCMS item-types which conforms to the concept. If the classification is documented in this way one can implement PCMS reports over the corresponding set of PCMS item types.

Inheritance

The Engineering Handbook classifies concepts in several dimensions, but does not organize the classification into an inheritance structure (generalization/specialization structure).

PCMS supports product typing, but not inheritance. APEL supports product typing and single inheritance. Thus, if a single inheritance structure is sufficient to represent organizational concepts, it can be described by APEL. The APEL inheritance structure cannot be directly reflected in PCMS. The inheritance tree depicted in Figure 4.6, page 70 was modeled, but was found not to be important for model understanding.
Aggregation

The only explicit use of aggregation in the Engineering Handbook was that the EH itself was modeled as an aggregate of its documents. In the repository this materialized as a part-id with an item-id for each of the documents.

PCMS supports type level aggregation by part-types (part categories). A part-type consists of a type-name and an optional list of item-types that should occur within the part. Item-types are not enforced, even when specified. PCMS supports instance level aggregation by part-ids. A given part-id may contain part-ids and item-ids of any number and any type. Attributes can be associated to both part-types and part-ids. The EH could have been modeled as an EH part-type, containing an EH item-type, or containing one item-type for each of the EH documents.

APEL supports general aggregate relationships, which can be used to create different aggregate structures. If such aggregates are created they should have a defined mapping to PCMS parts, or only refer to item content (e.g. sections in a document). The PCMS part structure is actively used by process agents to structure items, and if no mapping is provided the aggregates will be more confusing than helpful. The mapping must be maintained manually, because there are no mechanisms in PCMS to enforce properties on parts of a given part-type.

Association

The Engineering Handbook did not use any explicit relationships between product types, other than through activity inputs and outputs. PCMS does not support general associations\(^1\), only the predefined usage relationship among parts and between parts and items. The main purpose of this relationship is to be used in the configuration build process, supported by PCMS. In this context the relationship is used to select items and revisions to be included in a given configuration.

APEL supports general associations, making the modeler capable of creating association types. The created association types can then be used to relate product types in a product model. Associations differ from inheritance and aggregation in that they have user defined semantics. User defined associations cannot be interpreted by PCMS.

The role of product modeling

This section summarizes our experiences with APEL product modeling within the context of the studied case. We are asking the question: What is the purpose of APEL product modeling in the current Company practices, i.e. where the Engineering Handbook and the PCMS system are in use.

APEL provides many product modeling concepts and this can be applied to create complex product models. APEL product modeling must be seen as analogous to the PCMS control plan. APEL does not support any instance level product modeling, e.g., part breakdown structures cannot be modeled. The experiences with APEL product modeling is that it is of

\(^1\) The term “association” is used as in OMT, it resembles the relationship in an ER-model.
limited value since it cannot be supported by PCMS. Furthermore, there are product modeling concepts in PCMS not supported by APEL, e.g., product lifecycles.

To be valuable in the studied context, developed APEL models must be mapped into PCMS control plan concepts. However, since APEL does not support a superset of PCMS concepts, automatic translation cannot be achieved. Thus, APEL must only be seen as supplementary to PCMS control plan definitions. Since the product model cannot be supported by PCMS, the developed product model must only be seen as graphical and textual support to process agents. In this perspective, both the developed product model and the mapping to PCMS concepts must be placed in the Engineering Handbook. Furthermore, the relevant subset of APEL concepts are also a subset of OMT modeling, and the Company can substitute the APEL product model with a commercial OMT tool without losing any functionality.

From the perspective of the process modeler, there is a potential benefit from the tool support. The potential support is checking of consistency and correctness between the activity model and the product model. This was not supported by the APEL V2 tool. However, it should be rather easy to implement that each product variable in the activity models has a defined type in the product model. Thus, assuming that the activity model can be used by the company, this would be a benefit over using a separate OMT tool.

5.2.3 Product lifecycles and revisions

The product lifecycle is an important concept within PCMS. The lifecycle decides when historical information is produced (action item), and when item revisions are produced (extract item). In PCMS, a lifecycle is defined independently of item-types. A defined item-type must be associated to exactly one lifecycle.

The role of product lifecycles and revisions is discussed in terms of criteria for choosing lifecycle complexity, criteria for associating a lifecycle to an item-type, and the influence of activities on the choice of lifecycles.

Lifecycle complexity

The PCMS lifecycle is a user-defined state transition diagram. The states are freely defined, but transitions must be previously defined roles. PCMS requires that a lifecycle must have a normal path from an initial state to a final state. The normal path has specific semantics when PCMS*CTS is in use. The role of lifecycles in PCMS*CTS is discussed in Chapter 6, here we limit the discussion to the cases when PCMS*CTS is not in use.

The simplest possible lifecycle is two states, the initial state and the final state, and a transition between these two states. A created item will enter the initial state. A PCMS item in its initial lifecycle state can be extracted and returned without creating new item revisions. If an item revision is in any other state, an extraction of that item causes a new item revision to be created. Thus, the simplest possible lifecycle can be thought of as a “work” state, followed by a transition into its “validated” state.

The criterion for choosing lifecycle states is the number of distinct “validation” states that may follow a single “work” state. Used for real validation purposes, each “validation” state
should have a positive and a negative outcome, and a negative outcome should imply that a
new item revision is to be created. The “validation” states can also be used for trace purposes,
e.g. to indicate a “wait” state. In this case there do not need to be any alternative outcomes,
it is only a mark to show that a particular situation has occurred, or that a specific activity
takes place. The use of an explicit state is useful for project status reporting, furthermore,
PCMS stores information about all state transitions, and this information can be used in
reports and analysis of historical information.

PCMS offers functionality to overrule the state transition diagram, by allowing the product
manager to action an item from any current state, to any valid state. The intended use is
to fix accidental state transitions, e.g., when a process agent has erroneously actioned an
item to the wrong state. This functionality was “misused” by the Company in the following
way: In the Document review process the documents to be reviewed were actioned into state
DRAFT. Thus, if the review process had a negative outcome, one should create a new item
revision for each of the rejected documents. Instead, the rejected documents were actioned
back to their initial state (UNDER-WORK). In this state the documents can be updated
without producing new item revisions. The purpose of this “misuse” is to save disk space,
and to have fewer item revisions in the revision chain. The negative consequence is loss of
historical information, both with respect to item content and lifecycle information. Lifecycle
information is lost because PCMS only has one historical entry for each transition, and when
the item is actioned back to its initial states, all information connected to old transitions is
removed. This “misuse” is put in quotes because although it is a misuse of the intended PCMS
semantics, it is not necessarily a misuse with respect to Company needs.

**Lifecycles and item types**

In principle, each PCMS item type can have a unique lifecycle. However, this would be rather
confusing for the process agents, and would not utilize the fact that item-types and lifecycles
can be independently defined. In the studied case, all documents applied one of four lifecycles,
namely SIMPLE, SW-GENERATED, DOC-1, or SW-SRC. However, SIMPLE and SW-
GENERATED are identical (two states), and DOC-1 and SW-SRC are also identical (four
states). We can categorize them into two groups:

- **Generated documents.** These documents are derived files, generated by software tools.
  It has no meaning to update these documents, thus, it has no meaning to review these
documents, e.g. traces of software executions. These documents applied the SIMPLE
  or SW-GENERATED lifecycles.

- **Reviewed documents.** These documents are, or can be, subject to the document review
  process. These documents apply the DOC-1 or SW-SRC lifecycles. Typically, system
documents are subject to review, administrative documents are not. The two subgroups
can be characterized by:
  
  - System documents. Requirements, design documents, and user documentation
    written in a word processor, documents generated by the CASE tool, or source
    code (if not derived).
- Administrative documents. Engineering handbook, project plans, quality plans, etc. These documents can be subject to the document review process, but the reviews are either performed with less regularity or in some cases, not at all.

Some administrative documents such as forms, checklists, minutes of meeting, etc. will typically not be subject to a formal review process.

**Lifecycles and activities**

Lifecycles are associated to item types, and each item-revision follows the lifecycle specified for its type. The change of lifecycle states are done by action item commands. Action item commands are performed by process agents within the context of activities. A given item revision is related to activities in the following way.

1. *Creation – Systems development.* A document instance is created and enters its initial lifecycle state (UNDER-WORK) in one of the leaf-level systems development activities.

2. *Feedback – Document review.* The document instance is submitted for a review and enters its next lifecycle state (DRAFT). For a given document this may have two outcomes, either the document enters its REJECTED state or it remains in its DRAFT state. In the latter case the document will have a review status equal to OK when the document review process ends.

3. *Approval – Phase ending.* The document instance is at some point APPROVED by the project manager, if not before, at the end of the development phase in which the document was created.

4. *Rework – Change management.* If an error is found in a document that has been APPROVED it can only be changed by means of a change management process. A change management process takes an APPROVED document revision, and produces a new APPROVED revision. The new item revision is also here APPROVED by the project manager.

By looking at the lifecycle state of an item revision with the DOC-1 lifecycle, we cannot differentiate between a document which is in the review process and one that is ready for approval. Furthermore, when the document is actioned back to its initial state, we do not know that the review has taken place (in terms of revisions and the item revision history). The consequence is that PCMS reports show very little information about the processes a given document has been part of.

![Figure 5.2: Alternative DOC-1 lifecycle](image-url)

122
Figure 5.2 depicts an alternative to the DOC-1 lifecycle. With this lifecycle the document can be in a document review process from DRAFT to FINISHED (ready for approval). The document can be actioned from UNDER-WORK to DRAFT before entering the review process. The outcome of the review process is either ACCEPTED, FIX-REVIEW, or REJECTED-REVIEW. Documents accepted by review can either be APPROVED by the project manager, or be REJECTED-APPROVAL. Documents with REJECTED-REVIEW must be part of a new process, while those with FIX-REVIEW must not.

If item revisions are never actioned back to their initial state, one has information about cycles. E.g., in the case of a FIX-REVIEW a controller may action the item from DRAFT to ACCEPTED, or from DRAFT to a new FIX-REVIEW. Since a new revision is created each time, the number of DRAFT – FIX-REVIEW revisions will show the number of iterations.

The lessons learned is that to benefit from historical repository information, one should carefully design lifecycles, particularly with respect to the start and end of development processes. Furthermore, one should be less concerned about creating new revisions, and more concerned about the historical value of an item revision.

5.2.4 Choosing product abstractions

This section summarizes experiences with the use of product abstractions, independent of using the APEL product model or any other formalized approach.

It is useful to distinguish between the strategic and tactical aspects of product abstractions. A strategy is concerned with why abstractions are defined, i.e., goals beyond a single software project. For a given strategy one should develop the tactics for individual projects, i.e., how to use the abstractions in a given situation. Without a proper strategy for the use of PCMS concepts, the repository information has little value in a process perspective.

From a process modeling perspective the ideal situation is to use only PCMS item-types as activity input and output types. This ensures that all types have a direct manifestation in the repository. This has the following advantages:

1. Process modelers can be assisted by repository reports to ensure correctness and completeness with respect to the product types referenced by activities. Since PCMS does not include any activity information, this can only be done at the project level.

2. The process agents will benefit from a more operational process model, i.e., process agents can understand input and output types in the activity model from a PCMS perspective. This reduces the risk of mistakes and increases the likelihood of useful feedback to the process modeler.

3. Quality assurance personnel can be assisted by repository reports to assess process conformance. E.g., based on the knowledge of the end date of an activity and item-types that are to be produced by the activity, they can generate lists of PCMS items which existed at that date to check activity output conformance.

4. Process improvement personnel can use repository information. Valuable information includes counts of item instances in various categories to obtain volume information,
and to make time dependent product distributions.

While project participants and project management are mostly concerned about tactical aspects, other personnel are more concerned about strategic aspects. However, there are strong relationships between strategic and tactical aspects, in both directions, and these relationships must be maintained to achieve overall organizational goals. This means that strategic decisions must be realizable, i.e., practical considerations must be included. It also means that tactical decisions must conform to strategic decisions, and that strategic personnel must receive and listen to feedback from tactical personnel.

5.3 Human resources

This section discusses the modeling of human resources on the background of our experiences with APEL modeling of the roles found in the Engineering Handbook. The following discussion will refer to the APEL role modeling concepts described in Section 4.4, page 72. Furthermore, I will refer to the Engineering Handbook role descriptions and the corresponding APEL model described in Section 4.4, page 72.

This section is organized as follows: First, I discuss the problem of different semantics associated to the role concept. Second, I discuss role semantics defined in terms of role–agent allocations. Third, I discuss role semantics in terms of activity–role bindings. Finally, I discuss the team concept and its semantics with respect to agents, roles, and activities. At the end each section I summarize my experiences by proposing some properties that should be present in a model of human resources.

5.3.1 The role concept

By studying the roles used in the Engineering Handbook I found that no consistent semantics could be identified. The concept of a role was used to denote many different aspects of a human resource, this can be illustrated by the following examples:

- **Designer:** Used to denote a capability to design software by means of SDL.
- **Project manager:** Used to denote a formal position within the organization.
- **Author:** Used to denote a relationship between an agent and a document.
- **Controller:** Used to denote a temporal relationship between an agent, a process step, and a document.

Also, it is often implicit that certain roles are unique to a single agent in a given context, e.g., there is only one project manager within a given project. In other cases it is often implicit that any agent can play a certain role, e.g., author. If a single role occurs several times within a process, it is often implicit that the same agent should play that role. On the other hand, different roles within a process are often used as a suggestion to allocate different agents.
5.3.2 PCMS roles

As used by the Company, the roles defined within PCMS were only used to implement access control on products. Thus, there are few relationships between EH roles and PCMS roles. The DOC-1 lifecycle has two roles, author and reviewer. All project members were allocated to the author role, while only the project manager was allocated to the reviewer role. In a way, the author role is the same role as in the Document review process, in the sense that the initiator of a Document review process must be allocated to this role. However, the PCMS author role is not connected to a particular Document review process, which is the meaning of the EH author role. The PCMS reviewer role corresponds to both the EH project manager and change manager, both played by the project manager.

This confusion can either be caused by lack of a Company strategy for using PCMS roles, or by mismatch between the PCMS role concept and company needs. The most likely explanation is the latter, because PCMS roles are connected to products, not processes, while in the EH there is a need to connect roles to processes. A related issue is that PCMS roles are static in the sense that an agent is allocated in the part hierarchy. This is most useful to static project roles, e.g., project manager, method responsible, change manager. Dynamic roles that frequently change with respect to the allocated agent, must in practice be allocated to all project members, e.g., author. In this case the role concept loses its meaning because all these roles can be replaced by a list of project members.

5.3.3 Roles and agents

The implicit semantics of an APEL role model with respect to process agents is that an agent can be bound to role variables (of type "role type"). In the case of role multiplicity, several agents can be bound to the same role variable. However, in APEL there are no modeling concepts associated to agents, thus, individual agents cannot be modeled, neither can their relationships to roles be modeled.

Our experiences with the APEL approach is as follows: Roles without an interpretation in terms of agents do not contribute to increased process understanding. The only contribution of the APEL role concept is the implicit interpretation that a role variable occurring twice within an activity is supposed to be played by the same agent. This is only one of many possible interpretations that can be made in terms of agents, there are many other aspects of the agent-role allocations that are essential to process understanding. In the following we will give some examples from the modeled case in which the concept of an agent is needed to express the meaning of a role. In most cases the meaning can be sufficiently described by typed predicate logic.

**Example:** The Project manager role is always played by a single agent. □

**Example:** The Change responsible role in the Change Management process is bound to the agent that plays the Project manager role in the project. □

**Example:** The Reviewer role in the Document Review process should be bound to 2–4 different agents depending on the importance of the documents that are to be reviewed. In this case we need to express min and max cardinality, and that agents must be different. □
**Example:** More complex examples can also be found. E.g. in the Engineering Handbook explanation of the Document Review process we find the following statement: “It is preferable that the Author does not play the Manager or the Secretary role. The Author shall not play the Controller role.” □

Statements involving terms such as preferable cannot be translated into typed predicate logic. To express the meaning of such statements one can use deontic logic.

**Example:** The last sentence in the example above, “The Author shall not play the Controller role” can be specified by typed predicate logic. □

**Example:** In the last example there is an implicit assumption of a single process instance. The agent being Author in one instance (Review-1) cannot play the Controller role in the same instance, but can perfectly well play the Controller role in another instance (e.g. Review-2). □

The examples show that a more precise semantics can be stated when we introduce the agent concept. Furthermore, such information is essential to process management, quality assurance, and process improvement. E.g., it is important for all these activities to know that the author should not control his own documents in the review process.

<table>
<thead>
<tr>
<th></th>
<th>PRses</th>
<th>PRman</th>
<th>MRes</th>
<th>AnaT</th>
<th>DesT</th>
<th>Des</th>
<th>ImpT</th>
<th>TesT</th>
<th>TSpe</th>
<th>TRes</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProInv</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>QuaMan</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ProMan</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Analys.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SysDes</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DesImp</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Integr.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

Table 5.1: Activities – Roles and Teams

### 5.3.4 Roles and activities

Table 5.1 shows the relationship between project phases (rows) and role or team types (columns). The X means that the team or role is associated to an activity within the corresponding phase. In this section we will discuss the roles, the teams will be discussed in the next section.

There were three project roles that were not part of a team or part of the change management or document review processes. The roles and their explanation are:

- The project manager denotes a single person within the the software development group. Usually, this agent uses all his time on a single project. The project manager is a resource in the Project Invocation phase, and the only resource in the Project Management activity.

- The method responsible role denotes a single person within the software development
group. This person is the method responsible for all projects within this organizational unit. This agent is one of the resources in the Project Invocation phase, and the only resource in the Quality Assurance activity.

- The project responsible denotes a single person outside the software development group (on a higher organizational level). It is one person for each project, but the person may vary from one project to the other. This agent is one of the resources in the Project Invocation phase.

Seen from the project viewpoint, all these roles are played by a single agent throughout the project period. Thus, each time one refers to the role within a project, one also refers to a single person. This is generally not true for the other roles.

All roles in the Change Management and Document Review processes are defined relative to these processes. That is, these roles are defined by the process in which they occur, and they have no meaning outside this process. Furthermore, any project participant can play any of the roles in these two processes. The bindings between roles and agents usually occur when a given process instance is started, and then the bindings are fixed until the process terminates.

There is only one exception to this general rule, the change responsible defined within the Change Management process. This role is different from the other roles in two ways. First, this role is allocated to a single agent throughout the project period. This agent plays the change responsible role in every instance of the Change Management process. Second, this role is normally bound to the same agent as is bound to the Project Manager role.

APEL provides only one role model, where the types are globally visible in all activities. There is no way to define role types within activities and sub-activities. We find two problems with this approach: First, in the case most roles were defined relative to an activity, thus, a single global model is conceptually wrong. The consequence is lack of understandability. Second, the global role model causes unnecessary and meaningless work when an activity is to be reused in a new context. To reuse activities from another project model one needs to add or redefine existing role types in the role model. This is meaningless when the role is only used in the newly introduced activity.

In the studied case I find that the role concept is very context sensitive, the context being the activity in which the role is used. Rather than harmonizing role names in the activities one should have a separate specification of the role-agent binding associated to each activity. This experience can be formulated in the following way:

"It should be possible to define roles in the context of a single activity. There should be a separate specification of the role-agent binding. The role-agent binding should be able to refer to roles in another activity context."

**Example:** We should be capable of expressing that the change responsible is a role type within the change management process, and project manager is a role type within the project. Furthermore, we should be capable of expressing that the agent allocated to the project manager role in the project context should be identical to the agent playing the change manager role in the change management context.\[1\]
5.3.5 Teams and activities

In this section we discuss the relationships between teams and activities. We will use the term phase as it is used within the Company. Phases are a specific set of project activities, here we only consider a subset, namely Project Invocation, Analysis, Design, Detailed design and implementation, and Integration. For these activities there are only one activity instance within the project.

Table 5.1 (page 126) shows the relationship between project phases and project teams (AnaT, DesT, ImpT, TesT). We observe the following:

- The *Analysis team* is only associated to the *Analysis* phase. The other three teams are associated to more than one phase.
- The *Design team* is associated to the *System design and Detailed design and implementation* phases.
- Both the *Implementation team* and the *Test team* are associated to the *Detailed design and implementation and Integration* phases.

Although different teams were associated to activities, they were not explained in the Engineering Handbook.

One interpretation of Company teams is that people are specializing on different aspects of the project. The specialization is reflected in the defined teams. The specialization aspects are partially overlapping with the development phases. E.g., there are test activities within the *System design phase* (the Test specifier role in Figure 4.14, page 82).

Another interpretation is that the team is only used to specify that more than one agent is associated with the activity. This cannot be expressed by role multiplicity if these agents can play any of the roles in the activity. This interpretation is supported by the lack of defined roles within teams, e.g., only the test team has more than one associated role. Furthermore, project members participated in all teams, and there seemed to be nothing in the team concept that was not a consequence of the activity to which it was bound.

We find that one should separate between the set of people working on an activity or a task and the concept of a team. This experience can be summarized as follows:

- It should be possible to define activity resources without defining the resources as a team. The activity resources should be the agents allocated to the activity, defining the set of agents that can be allocated to roles within the activity.
- A team should be defined within the project, independent of specific project activities. It should be possible to specify a team as an activity resource. The binding between activity roles and team members should be done within the activity (similar to the binding of activity roles to the roles of the other activities).

The discussion in the previous section about roles and agents is also valid for teams. E.g., it should be possible to specify the agents allocated to roles within a team, that an agent cannot be the member of two teams, or that two teams cannot have the same members.
5.4 The process modeling environment

This section describes experiences with the APEL modeling environment, with respect to the needs and use within the studied case. I limit the description to overall, conceptual issues and do not go into detail about tool functionality.

5.4.1 Process modeling languages

There is an intuitive difference between a language and its supporting environment. However, when the language is not formally defined, the supporting environment becomes the de-facto language. This has the unfortunate consequence that if the environment does not work, there are no language. This is a particular problem with research prototypes, because they often have missing functionality or features that do not work properly. This was also the case with the APEL language and environment. The lessons learned is that formal languages must be formally defined. Furthermore, one should separate between properties of the language and the environment.

- **Formal languages.** The lesson learned is that formal languages are useful for process modeling, but that a formal language should *not* be able to express every possible situation that might occur. A formal language should include few, well defined, and well understood concepts.

- **Diagrammatic languages.** The lessons learned is that diagrammatic languages are useful for model understanding, but that one must limit the amount of information in a diagram. Overloaded diagrams are worse than the corresponding textual description, e.g., more difficult to understand and more tedious to update.

The main problem with APEL is that it has ambitions about both being a process modeling and process programming environment. Our experience is that process modeling is fundamentally different from process programming. Process modeling should be concerned about understanding and describing company processes, regardless of programming aspects. It is not desirable to mix modeling with programming.

5.4.2 The product repository

APEL provides many product modeling concepts and this can be applied to create complex product models. However, the experiences with the APEL product modeling is that it is of limited value since it cannot be supported by PCMS. To be utilized, abstractions not supported by PCMS must have a defined mapping into PCMS supported concepts.

5.4.3 Generic and specific models

APEL assumes that there is a one-one correspondence between the generic description and the performed process. The existence of generic (GEH) and project specific descriptions (EH) within the Company invalidates this assumption. There are two important implications:
1. Modeling the generic process has little relevance for the performed process, if not supplemented with modeling of project specific processes.

2. Maintaining a generic description and a set of project specific descriptions causes a new set of requirements on feedback and consistency. At the same time, project specific models (EHs) must be allowed to be inconsistent with the generic model (GEH), and the generic model must be updated by experiences in individual projects.

The company needs to update the generic model and the respective project models concurrently, possibly by different users. The collection of APEL models are stored in one big file that cannot be split. There might be any number of such files, but it is not possible to share any information across these files. If APEL were to be utilised within the company there must be one versioned APEL model (.gti file) for the GEH, and one for each of the EHs. APEL does not provide instantiation support beside a complete copy of a .gti file, and there is no feedback support. Essentially, APEL is not modularized, and thus, does not provide even the simplest export and import functionality. Several users might open editors on a single .gti file and edit any model, but APEL does not provide concurrency control.

5.4.4 The process manual

The APEL environment assumes that the process agents use the tool for process look-up and understanding. The interest of the organization was to use the environment for the purpose of making the models, and then include the models in the Engineering Handbook. The major reason for this is that it is more convenient to read printed process documentation. Furthermore, the process documentation is also to be used by people that cannot be expected to know the process modeling tool, e.g., external quality audits, customers, and high level management. Thus, the models must be documented in the Engineering Handbook.

**Example:** The activity input and output is displayed as product variables, while the product model displays product types. Thus, the process agent cannot see the relationship between the activity model and the product model. □

To summarize, a major requirement towards process modeling environments is to have powerful documentation support. They must either have the capability to replace the existing process Engineering Handbook, or be tightly integrated with a word-processor.

5.5 Types and instances

Research in *process modeling* has focused on how process engineers can create type-level models, largely ignoring the need for instance-level modeling. On the other hand, research in *process enactment* has focused on how instance-level models can be automatically created from the type-level description, ignoring that it is often a creative process that cannot be automated.

The idea of types and instances is well known, e.g., from object-oriented programming languages. Here we require that both types and instances represent concepts in the modeled
organization. The difference between types and instances can be defined as a difference in their possible occurrences:

1. **Type.** A type represents a concept that can have several occurrences in the organization.
2. **Instance.** A instance represents a concept that has at most one occurrence in the organization.

The instance definition must not be misunderstood as the presence of an objective one–one correspondance. What is meant is that the people in the organization know the mapping. E.g., a development phase is an intangible concept, but people in the organization can easily identify a unique occurrence within a company project (e.g., the analysis phase within PRO1 is unique).

![Diagram](image.png)

**Figure 5.3: Types and instances**

Figure 5.3 illustrates the idea of types and instances in terms of four models. The figure shows that types may have instances, and that instances may have types. It also illustrates that types refer to types, and that instances refer to instances.

1. **Process types.** A1, A2, and A3 are process types. The arrows denote finish–start dependencies between the process types. Process types refer to product types, e.g. A1 consumes a product of type B2 and produces a product of type B2, etc.

2. **Process instances.** T1, T2, T3, and T4 are process instances. The arrows denote finish–start dependencies between the instances. Type A1 has instance T1, A2 has instances T2 and T3, and A3 has instance T4. In the opposite direction we say that T1 has type A1, etc. Process instances refer to product instances, e.g. T1 consumes p1 and produces p2 and p3, etc.

3. **Product types.** B1, B2, and B3 are product types. The lines denote that type B1 may contain one or more products of type B2 and B3.
4. **Product instances.** p1–p6 are product instances. The lines denote that instance p6 contains the instances p1–p5. Type B1 has instance p6, B2 has instances p1–p3, and B3 has instances p4 and p5.

Furthermore, we can say that, e.g., T2 and T3 are consistent with A2 since T2 and T3 consume and produce products that are conformous to the definition of A2 (p2 and p3 have type B2, p4 and p5 have type B3).

Different kinds of systems will provide support for different aspects. The APEL V2 environment supports both product modeling and activity modeling. However, both are supported on the type-level, none at the instance level. Configuration management systems such as Clearcase support product types and product instances. Project management systems such as Microsoft/Project support activity instances. If we include the PCMS*CTS functionality, PCMS provides functionality in all the four categories, as illustrated by the following example.

**Example:** Plain PCMS provides product modeling support, and PCMS*CTS provides activity modeling support. Both product modeling and activity modeling are supported by both types and instances, e.g., item-types (product type), items (product instance), change document types (activity type), and change documents (activity instance).

In the case of PCMS product instances, one can easily be lead to believe that the repository is not a model, but the actual product. In a way this is true, particularly when the final delivered software is contained in, or generated from the repository. Nevertheless, it is more useful to see the repository as an instance level product model, than being the product itself. There are two main reasons for this: First, the repository is rarely delivered to the customer. The repository contains additional information beside the delivered product, e.g., the design structure, historical information, and access control information. Second, it is not true that everything is inside the repository, e.g., printed documents, manuals, and equipment are not part of the repository.

The process modeling literature uses the term *model* as being synonymous to type level model. The rationale behind a *process modeling language* without instance level models must be either that there are no differences between type and instance level models, that there is no need for instance level models, or that researchers simply limit the scope to type level modeling. The former two positions will be discussed in the following sections. The latter must be accepted as a valid approach, but the consequence is that only the kind of information found in process manuals can be modeled.

### 5.5.1 Type level models

By type level models we mean information about process types (activities), product types, and categorization of human resources. Type level models include relationships among type level concepts, e.g., that an activity is related to a product type by a product input relationship. In the following we will limit the discussion to how activities and products should be related in a type level model, i.e., to use products to denote both data and control flow.

The Company use availability of documents in a specific state as their main control concept. In the studied processes a data-flow was either the control-flow or some background information.
Moreover, a control flow that does not correspond to a document, or information about a document, can be ignored in a type level model. This because there is too much uncertainty associated to such control information.

A preferable style of diagrams is depicted in Figure 5.4. This diagram depicts an activity network (process-type) with four activities. The diagram depicts three relationships, namely A–B, A–C, and A–D, called document transportation. The small circle is not a decision point, it is only used for graphical convenience. Document types are distributed by human resources in activity A, to human resources in activities B, C, and D. The precedence interpretation of the document transportation can be any of start-start, start-finish, finish-start, and finish finish. The only limitation is that a strict start-start dependency cannot be modeled, this because we require that Activity A produce at least one document of a type which corresponds to the input type of either B, C, or D, before the succeeding activity may start. Figure 5.4 can be explained as follows:

1. An activity may produce documents that are not transported by this process (e.g. Activity A, DOC4).

2. Three documents are transported (DOC1,DOC2,DOC3), i.e., used as control signals for a set of activities (B,C,D). Activity B and C only consider documents of type DOC1 and DOC2. Activity D only considers documents of type DOC3.

3. Starting conditions for activities must be specified using the enter criterion of the various activities. The enter criterion of an activity must refer to attribute values (e.g. PCMS status) in one or more of its input documents. E.g. that Activity B may start when DOC1 from Activity A has a status equal to APPROVED.

4. An activity may consume documents that have not been transported by this process (DOC5, Activity D), thus not considered as a control signal.

The diagram does not tell you whether Activity A is executed concurrently with Activity B, C, or D. Neither does it tell whether Activity B, C, or D are alternatives to each other or that they should all execute. However, this is often obvious to the personnel involved. If they need
to check, the answers can be found by reading enter and exit criteria. In the Company case, the enter and exit criteria can be specified by logical expressions over PCMS parts, items, life-cycle states, and attribute values.

Note that the diagram depicted in Figure 5.4 can be simplified by removing redundant information. The only information that must be present is the relationships between pairs of activities and the input and output lists of the respective activities.

### 5.5.2 Instance level models

By instance level models we mean information about activity instances, product instances, and instances of human resources. Instance level models include relationships among instance level concepts, e.g., that an activity instance is related to a product instance by a product input relationship. Instance level models also include information about entities and relationships, e.g., attribute values.

While a type level model is (or can be) developed before the project starts, the instance level model is created as part of the project execution. A type level model can be independent of any project, while an instance level model is always developed in the context of a specific development effort. However, an instance level model can perfectly well be projective, stating the desirable future, e.g. in the form of a project plan. Furthermore, an instance level model can possibly say something about the current state of affairs and about historical events. Thus, instance level modeling is a large topic, but the following discussion is limited to the relationship between activities and products, more or less as depicted in Figure 5.3. While we previously identified the activity–product, and type–instance dimensions, we will here illustrate the problem of deciding the instantiation semantics.

![Diagram](image)

Figure 5.5: *Instantiation and binding*

Figure 5.5 illustrates the one–many instantiation and binding problem. The type level model contains two activities (A and B), related by a transportation of two document types (X and Y). The instance level model contains one instance of A, namely a1, and two instances of B, namely b1 and b2. Here, a1 is related to both b1 and b2, with two document transposition relationships. One can say that the instance level model is conformous to the type level model because A’s are followed by B’s, and the transports are X’s and Y’s.

We can easily identify a large number of alternative semantics and possible distributions of X and Y instances, e.g., must both b1 and b2 receive documents of both types, can b1 receive all instances of X while b2 all instances of Y, are all instances of X and Y valid in this context, must the sets of document instances be partitions? etc. Still, this is the simple case,
a more complex case is when there are many instances of both A and B. This is illustrated in Figure 5.5, denoted as the many-many instantiation and binding problem.

The key point is: These situations exist in the organization and represent a modeling problem which cannot be resolved by APEL because APEL V2 did not provide sufficient instantiation and binding semantics in the activity multiplicity constructs. What is more important is that most of these problems cannot be resolved by increasing the modeling power, for the simple reason that there is no best strategy. Binding and instantiation depends upon the situation, e.g., who wrote a document, how big is the work to be done, how many people are available, what are the project priorities, which documents are currently being edited, etc.

The motivation for proposing an explicit instance level model is twofold: First, I experienced that instance level information was too project specific to be included in the type level model, i.e., it is subject to too many unknowns, uncertainties, and exceptions. Second, an instance level model is clearly useful. With an instance level model one can express more exact process properties, thus, the model becomes more operational and correct. As the project moves forward one knows more and more about instance level issues, e.g., at a certain point one knows which documents are allocated to a task, and the personnel working on that task. Process performers are more directly supported, because models are less abstract, i.e., an input item can be found in the repository. Furthermore, an instance level model may replace project management software and thus have an immediate pay-off and an operational use, i.e., planning and progress reporting. Finally, An instance level model contains the actual development history for this particular project, information that can be used in an inductive way, e.g. for quality assurance purposes, for process improvement purposes, or for process modeling purposes.

5.6 Summary

This chapter has reported experiences in process modeling. The work is based on the company Engineering Handbook, the company use of PCMS, existing PCMS product structures, and interaction with company personnel. On this basis I developed APEL models of company activities, products, and resources. The experiences reported in this Chapter can be summarized as follows:

Single activity. A single activity has been discussed in terms of its inputs and outputs, enter and exit criteria, and multiplicity. The experiences are that input and output must be modeled as document types, and enter and exit criteria as logical expressions over document lifecycles and attribute values. With respect to documents one should separate between documents as stored in PCMS and print-outs; the flow of paper copies is important with respect to understanding the development process. Activity multiplicity is generally unknown or undecidable, and must only be used to indicate typical situations.

Activity chaining. Activity chaining has been discussed in terms of the distinction between data and control, and the use of chaining concepts such as sequence, parallelism, choice, and repetition. The experiences are that the distinction between data and control was not useful, it is preferable to use data as control. The dominant precedence dependency between activities is the finish–finish dependency, not the APEL finish–start dependency. Both can be expressed by
logical expressions over properties of input and output documents. In a real project situation, sequencing depends upon resource availability, previous decisions, and priorities. Thus, the sequencing defined in the activity model will always be subject to uncertainties. Activities modeled as parallel are not necessarily independent, i.e., in some cases there might be a need for extensive communication between process agents working on parallel activities. Parallel activities should be modeled when they can be identified as having responsibility for different document types. Multi-instance activities should be modeled when they can be identified as having responsibility for different document instances. Choices can be modeled as conditions over document properties. Repetition should be divided into feedback and rework, feedback when the activity instance is active, rework when the activity is finished. Rework should only be performed by new activity instances.

**Activity decomposition.** Activity decomposition has been discussed in terms of properties of leaf-level activities, properties of detailed activities, and the reasons for differences in the level of detail. The experiences are that a detailed activity is best understood in terms of operationability, i.e., to what extent a process “freshman” can perform the process solely on the basis of its description. Three important properties of a detailed activity are few agents, few documents, and short duration. Correctness is defined in terms of the lack of difference between the process description and the actual process. An operational description is not necessarily correct, and a correct description is not necessarily operational. The reasons for differences in level of detail can be explained by needs and effects. Needs are classified as knowledge needs, quality assurance needs, and improvement needs. Effects are classified as understanding effects, coordination effects, certification effects, and management effects.

**Product modeling.** Product modeling has been discussed in terms of the difference between the EH product concepts and their manifestation in the PCMS repository, the usefulness of product modeling in a situation where the product model cannot be represented in the repository, the role of product lifecycles and revisions, and criteria for choosing product abstractions. The experiences are that process modeling and configuration management systems must be harmonized with respect to their respective product models. It is of little use to have an advanced PM product model which cannot be supported by the existing CM system. The two models should not be different, if they are different there must exist a mapping between the concept and its repository manifestation. Product lifecycles must be chosen on the basis of existing company processes, and must involve a strategy for creating new item revisions. Lifecycle design must consider the total set of activity types and product types because a product type can be used by many activity types, and an activity may use many product types. The choice of product abstractions should be made according to strategic goals, and according to needs of different categories of personnel, e.g. project participants, project management, process modelers, quality assurance personnel, and process improvement personnel.

**Role modeling.** Role modeling has been discussed in terms of different semantics associated with the role concept. The experiences are that too little semantics are built into the concept, and thus it is of little value. The role and team semantics should be defined in terms of agents and activities.

**Process modeling environments.** The APEL process modeling environment has been discussed in terms of its potential use within the studied case. The issues are the relationship between the modeling environment and the process manual, the role of formal and expressive process
modeling languages, and the role of diagrammatic process modeling languages. The main experiences is that a “type level” PME acts as production support for – or replacement of – the process manual.

*Types and instances.* Process modeling could be simplified, and made more useful, by separating between type and instance models. Types to model the relatively stable aspects, instances to model aspects which are different in each project or other execution contexts. The problem in process modeling is not so much to describe a “known” process, but to “know” the process. An instance level model, as exemplified by the PCMS repository, provides information on the actual process.
Chapter 6

Programming company processes

My main contribution in this part of the thesis is to provide experimental results on the application of process enactment. Here I report a controlled experiment based on two processes identified in the case study part of the thesis, namely Change management and Document review. The processes were realized in two different process support environments, namely Process Weaver and PCMS*CTS. Furthermore, the processes were performed Manually to create a basis for comparisons. This chapter describes the two environments, the process implementation, and the six execution alternatives. The experiment is described in Chapter 7.

This Chapter is organized as follows: Section 6.1 explains why I selected the Change management and Document review processes for automation. Section 6.2 explains why I selected PCMS*CTS and Process Weaver as support environments. The following two Sections describe the two environments, Section 6.3 PCMS*CTS and Section 6.4 Process Weaver. Section 6.5 describes the process implementations and the implementation context. Section 6.6 describes how each of the six process execution alternatives were performed in the experiment.

6.1 Selecting processes to be automated

The Company processes are described in Chapter 4, presented as Systems development, Change management, and Document review. Systems development is an abstraction of a large activity hierarchy of which possibly sub-activities could be considered for automation. With respect to generalization of experimental results it would be an advantage to include as many processes as possible to get a representative selection of project processes. However, only change management and document review were considered as candidates for process automation. There are two reasons for this: First, the interests of the company were directed towards these two processes. Second, these were the only processes which were defined down to a level of strict chaining and PCMS lifecycle control. Activities of other processes were often overlapping and more vague with respect to relating lifecycle states to enter and exit criteria.

Both Change management and Document review were seen as critical processes by the Company. First, critical decisions about the product are made in these processes. Second, they are
coordination intensive, as each process involves a number of people and many documents. To reduce the risk of errors, the Company have carefully described these processes, and moreover, ensures that they are carried out as described. As a consequence, there is some administrative overhead associated with handling change and review reports and the documents they refer to. Although this overhead is necessary, it has some negative effects, e.g., it consumes effort, it takes time, and is seen as unpleasant work by the project participants. Thus, any technology that can remove or reduce the overhead is welcomed by the Company, provided that the benefits of a controlled process are maintained. Furthermore, although there are other potential benefits in process automation, the Company’s main interest was to reduce the administrative overhead, defined as:

[Administrative overhead] “The work associated with extracting and returning items to and from PCMS as well as creating and maintaining the change and review reports.”

We decided to include both change management and document review in the experiment. The advantage gained by selecting one of them would have been twice as many experimental runs in the available time (since they were considered to have about the same execution time). Furthermore, it would have been cheaper to implement one process than two. However, the two processes had different characteristics and we felt that the experimental results would be too specific if we selected only one process. We wanted results that to some extent could be generalized within this particular Company environment, and to defend such a generalization we were convinced that both processes were needed.

6.2 Selecting implementation environments

The choice of support environments was pragmatic; Process Weaver was selected because it was available within the context of the PERFECT project, and PCMS Change Tracking System (CTS) because it was already available within the Company PCMS installation.

A consideration for the experiment was whether we should implement the two processes in both of the available support technologies. We considered Process Weaver to be more representative for process automation technologies (PSEE’s) than the PCMS*CTS. For this reason, Process Weaver was our first choice. Since PCMS*CTS was available at our PCMS installation and because the company wanted to investigate its applicability, we also included this tool in the experiment. We felt that it would be interesting to compare the two technologies in an experimental setting, and that this could give us some additional insight relevant to Company needs. Furthermore, we felt that the comparison of the two technologies could give us some general insight in process automation technology, and perhaps allow us to generalize some of the results.

Since data on the administrative overhead was not available for the manually performed processes, we had to measure this in the same way as we measured the overhead for the automated executions. This was done as part of the experiment, denoted as Manual PCMS.
6.3 PCMS*CTS

The PCMS Change Tracking System (PCMS*CTS) is process support embedded in the PCMS system, but currently not used by the studied organization. PCMS*CTS uses the concept of a Change document which is a special type of PCMS document. A change document has a lifecycle that can be connected to ordinary PCMS documents by a set of rules (flags). It has the concept of input documents (affected), output documents (in response to), roles, and history. Each time a change document is opened, it will be generated in such a way that it reflects the current status of the process.

This Section is organized as follows: Section 6.3.1 explains the PCMS*CTS Change Document in terms of types, lifecycles, templates, and roles. Section 6.3.2 explains the objects that can be related to a Change Document. Section 6.3.3 explains the flags that can be specified in Change Document and item lifecycles, and how these flags are used by PCMS*CTS rules. Finally, Section 6.3.4 describes how Change Documents are used.

6.3.1 Change document

A PCMS*CTS Change Document differs from an ordinary PCMS document in that it does not exist as a file, but only as a PCMS data structure. While normal PCMS documents are extracted, edited, and returned, a Change Document is simply opened and closed. Each time a change document is opened, it will be generated in such a way that it reflects the current status of the process. The end-user can only manipulate the Change Document through a predefined user-interface.

Types and lifecycles

Although change documents are fundamentally different from items, they have some common properties. Similar to a PCMS item, explained in Section 3.4 (page 43), a change document has a type and a lifecycle. There are only two differences, the template and the rule flag:

- **Template.** While an ordinary PCMS item has an unformatted template file, a change document template has a specific PCMS defined format.

- **Rules flag.** While ordinary PCMS items have rule flags which describe how they are related to change documents, change documents have rule flags which describe how they are related to items.

The rule flags are explained in more detail in section 6.3.3, here we describe the definition and use of change document templates.

Templates

A change document template is defined in terms of change document attributes. Change document attributes are “variables” with an associated presentation format. Each time a change
document is opened the actual values are displayed according to the defined presentation format. The template designer may either use PCMS defined attributes or he may define his own attributes.

- **User defined attributes:** Values of user defined attributes are always entered by the process agents during process execution. Typical user defined attributes are: Change document title, priority, time and resource estimates.

- **PCMS defined attributes:** Values of PCMS defined attributes are typically generated from information found in the PCMS database, according to the change document relationships (see section 6.3.2). Typical PCMS defined attributes are: Change identifier, creation date, originator, affected parts, affected items, description, action messages, document history, and related change documents.

Defining a Change Document in terms of attributes is similar to defining a report in terms of a report generation language, e.g., one can define multiple-rows, and multiple columns which are filled with information when the document is generated. The main difference with respect to a database report is that a PCMS Change Document can be manipulated (e.g. relate items, update attributes).

Change Documents are actioned in the same way as PCMS items. Each time a Change Document is actioned the user may include an action description, according to a user defined sub-template for action descriptions. PCMS allows Change Documents to be retrieved with the attribute values at any action point.

**Roles**

In PCMS one can define two types of roles, static roles and dynamic roles. Both types are declared on a specific instance in the PCMS part structure. If they are to be visible in the whole product structure, they must be declared at the root of this structure (product-id), and not be redefined at lower levels in the structure. The difference between static and dynamic roles is that in the case of dynamic roles, no process agent is allocated to the role. The two types of roles and their usage can further be described as follows:

**Static roles:** A change document is always associated to one or more parts within the product (by default to the product-id). When it is associated to a part, the roles and the agents allocated to these roles are automatically bound to the roles defined within the change document lifecycle. Where more than one design part is associated with a change document roles are assigned by default to the users who are responsible for the first common parent in the associated parts.

In the case when there are many agents associated to a role (within a part) one can define one of the agents to be the leader. The leader definition only has consequences for change documents. If defined, then only the agent playing the leader role is allowed to update the change document’s attributes, and furthermore, action the change document to its next lifecycle state.

**Dynamic roles:** A PCMS dynamic role is declared in the same way as a static role with the difference that no process agent is allocated to the role. Rather than allocating an agent one
defines a “candidate list” of possible agents. In our process implementations the “candidate list” includes every project member. Since the roles are declared they can be used in change document lifecycles. If the roles are declared in the root of the PCMS part structure (the product), they are valid for any change document created within the product. These roles can be used as input-fields (attributes) of the change document, and agents can be dynamically allocated from the “candidate list” when the change document has been created. These roles are local to the change document instance, as another change document may allocate another agent to the same role.

6.3.2 Change document relationships

Every change document belongs to a specific product. Change documents can have relationships to parts, items, or other change documents provided they all belong to the same product. In general, these objects can be related and unrelated to change documents at any time (it can be controlled by rules). An entry is logged in the change document’s history each time an object is related or unrelated to the change document (operation, user, time). Here we mainly consider parts and items:

- **Parts.** Any part can be related to any change document in the same product. The PCMS user who creates a change document (originator) may relate it to any of the product’s parts. If no parts are specifically related, it is automatically related to the product. Later, this part can be unrelated and/or other parts can be related.

- **Items.** Items which are subject to change are related to the change document as **affected.** This means that a new revision will be required to implement the change. When an item revision is extracted from PCMS, the new item revision created from the extraction will be related to the change document as **in-response-to.** An item revision can also be related to the change document as **info,** meaning that it contains relevant information, but this item is not subject to change. When a product item is related as affected, its owning part will also automatically be related as affected.

Furthermore, any two change documents can be related if the control plan permits their types to be related. Rules can be defined between a change document and its parent and/or children change documents. However, this functionality is not used in this work, and therefore not further described.

6.3.3 Rules

The interaction between change documents and items is defined by **rules.** Rules are relationships between the lifecycle of a change document and the lifecycles of items. The lifecycle associated to a change document type can be seen as a process definition, while the lifecycle associated to an item is the product definition.

Item revisions may be related to a change document, if rules are either not active or not defined. An item can be related to a change document if rules are defined for both, but not if one has rules and the other not.

142
Change document rules

In the lifecycle associated to a change document type, one can define rule flags which describe how the change documents of this type can be related to items in general (not to specific types).

When rules are to be applied to a change document type, its lifecycle states are grouped into phases. The phase associated to a lifecycle state determines the “operations” which can be performed on the change document while it is in that particular state (e.g., relate item as affected, update attribute values). There exists nine predefined PCMS phases, the following explanation is adapted from the PCMS User Guide [115]:

• **HELD phase**: The change document is in this phase while the originator holds it for preparation. It is not visible to other users as long as it remains in the held list. This phase does not correspond to any lifecycle state because the document is not yet created.

• **CREATE phase**: When a change document is created it is automatically in this phase. The phase is automatically assigned to all normal lifecycle states which precedes the analysis phase (if any).

• **ANALYSIS phase**: The phase should be assigned to the normal lifecycle states during which implementation of the change is analyzed and planned (where items and parts are listed as affected).

• **WORK phase**: The phase should be assigned to the normal lifecycle states during which the change is actually implemented (where items are extracted, edited, and returned).

• **FROZEN phase**: The phase should be assigned to the normal lifecycle states (if any) which follow the work phase but may precede the final normal state. During this phase no update to the document is allowed except for adding action descriptions and updating attributes.

• **CLOSED phase**: This phase is automatically assigned to the final normal state. During this phase no document update is possible.

• **REJECTED phase**: This phase is automatically assigned to all final states except the final normal state. During this phase no document update is possible.

• **OFF NORM phase**: This phase is automatically assigned to all other states: namely, all those which are not in the normal lifecycle, and are not final states. Only attributes can be changed in this phase.

• **AN + WORK phase**: This composite phase may be assigned to all normal lifecycle states between the create phase and the frozen phase, as an alternative to assigning separate analysis and work phases.

The assignment of phases CLOSED, REJECTED, OFF NORM, and HELD to lifecycle states is automatic. The control plan therefore needs to specify only how the states in the normal lifecycle path are to be partitioned into the phases CREATE, ANALYSIS, WORK, and FROZEN, or into the phases CREATE, AN + WORK, and FROZEN. This is done by the modeler in terms of four flags which are associated to states in the normal lifecycle.
Item rules

In the lifecycle associated to an item type, one can define rule flags which describe how the items of this type can be related to change documents in general (not to specific types).

For an item revision to be related to a change document the control plan must specifically permit the two types to be related, and the change document must be in a phase where the corresponding relationship is permitted to be established.

The rules which may be placed on item types and their normal lifecycle states are denoted by the following flags:

- **Ch-doc-flag**: Can be associated to an item type. It means that an item of this type cannot be created in other ways than through a change document.

- **New-rev-flag**: Can be associated to any lifecycle state. It means that new revisions of that item cannot be created unless it is related to a valid change document.

- **Action flag**: Can be associated to a normal lifecycle state. It means that items of this type cannot be actioned to the next state, unless they are related to a valid change document.

- **Close flag**: Can be associated to a normal lifecycle state. It means that items of this type must reach or pass this state before change documents, to which they are related, are allowed to close.

These rules make it possible to specify when a change document should be applied to different item types, and furthermore guarantee that a certain state is reached before a change document is allowed to close.

### 6.3.4 Using change documents

Any user with a PCMS defined role can create a change document, the change document keeps the user-id in the originator attribute. The user selects the type of change document he wants to create. Certain attributes can be set at this point and parts may be related to the change document. The user may also enter a detailed description of the change by means of a user defined editor. The user then has a choice of registering the change document, holding it, or deleting it. When registered the change document is moved into its initial lifecycle stage and is assigned a change document number. A mail message is sent to the user(s) who has the next role to action the change document. The user can browse and print it from their pending list.

The users that may update a change document are those for whom the change document is pending. Updates may be further restricted by the use of the leader function. To update a change document is to:

- Relate parts, items, or other change documents, to the change document.
- Edit the detailed description, action descriptions, or attributes, of the change document.
As for updates, the users that may action a change document are those for whom the change document is pending. Actioning may also be restricted by the use of the leader function.

6.4 Process Weaver

Process Weaver is the commercial process modeling and enactment tool developed and marketed by Cap Gemini Innovation. It is a package containing several tools denoted as the Process Weaver environment [25, 26, 27, 28]. In the following we will describe the Agenda, the Work context editor and the Cooperative procedure editor.

This section is organized as follows: Section 6.4.1 explains the Agenda, the tool in which work-contexts appear during process execution. Section 6.4.2 explains the Work-Context editor, the tool which allows the process programmer to create customized work-environments. Section 6.4.3 explains the Cooperative procedure editor, the tool which allows the process programmer to create process programs.

6.4.1 Agenda

Process Weaver operates with an Agenda tool associated with individual process agents. There can only be one agenda for each Unix-user. Work appears on the Agenda as a work-context, and remains in the Agenda until the user presses an answer button in the work-context.

1. Start. An icon will show up in the Agenda each time the user receives a work-context. When the user clicks on this icon, a work-context shows up.

2. Do work. The work-context contains tools and information the user needs to do his assigned work, e.g., documents, word processors, spread sheets, compilers.

3. End. When the user finishes his work, he clicks his answer button to report to Process Weaver that he is finished.

A work-context can be delegated to other users, but the initial receiver will still have status as the work-context owner (consequences for access restrictions). The work-context owner will again receive the work-context when the delegated user presses an answer button.

The Agenda is also used as a launcher for the other Process Weaver tools. E.g., the Agenda can be used to specify that a work-context should appear in the Agenda at specific dates or periodically.

6.4.2 Work Context editor

A piece of work is manifested as a Work Context (WC) on the agenda, either started (sent) by the agent himself, by other agents, or by a cooperative procedure. A given work-context is asynchronous in the sense that the agent can perform actions within a work-context in any
sequence until he presses an Answer Button which causes the control to be returned to the sender.

The work-context editor is a tool for designing work-contexts. The designer may use the following work-context objects:

1. **Answer buttons.** Confirming the termination of the process step described in the work-context. There might be several answer buttons within a single work-context, e.g., terminate with success and terminate with failure.

2. **Event buttons.** Sends a predefined event on the global Broadcast Message Server (BMS). The event is named BUTTON.CLICKED, and it includes information about the access path and filename of the work-context file. Any application may listen for such events and take action when they occur. If needed, the receiving application can check the work-context file for additional information. The typical usage is to synchronize a cooperative procedure. An event button does not terminate the work-context, the work-context remains in the agenda.

3. **Selection lists.** A list of values is provided by the work-context sender. The work-context user may then select one or more of these values. The selection is made known to the work-context sender upon work-context termination. The typical usage is to select process agents (users) and input documents for subsequent process steps.

4. **Edit lists.** This type of list is similar to the selection list, but the user may here create and edit the list elements. The complete list is made available to the work-context sender upon work-context termination.

5. **Text area.** Displays a fixed text to the user. The text may contain variables whose values are provided by the work-context sender.

6. **Editable text area.** Displays a text provided by the work-context sender. The work-context receiver (user) may edit the text. The text is made available to the work-context sender upon work-context termination.

7. **Numbers.** Integers and float values. Similar to an editable text area, but with integer and float semantics.

8. **Documents.** Any number of documents can be defined within a work-context. Each document is represented by a graphical symbol, and an editor containing the document will appear upon user selection. Which editor to invoke is decided when the work-context is designed. The file containing the document to be edited is represented by a variable. At run-time this variable is typically bound to a Unix path and file name by the sending cooperative procedure.

9. **Tools.** Tools are similar to document editors with the difference that they do not need a document (file). Tools can either be synchronous or asynchronous. A synchronous tool will be terminated when the work-context is terminated, while an asynchronous tool is independent upon the work-context.
Initially, a new work-context provides an empty canvas, populated with work-context objects in a graphical “drag and drop” paradigm. Objects can be moved, resized, etc. When the objects have been placed on the canvas an associated property editor can be opened on each of them.

An example of a populated object canvas is depicted in Figure 6.1. Figure 6.1 shows two documents together with the variables that will contain their names at runtime ($doc_name and $cfr_name). Furthermore, the figure shows two text areas, one selection list, and one answer button.

At runtime, work-contexts are sent to users from cooperative procedures. The user controls the work-context until he presses an answer button, then control is returned to the sending cooperative procedure. The communication between a work-context and a cooperative procedure is done by means of variable binding.

1. **Send.** The sender may provide values to work-context variables, when this is the case the binding occurs after the work-context send operation, but before the work-context is presented to the user. If the work-context is sent from a cooperative procedure, the variables receive values from the cooperative procedure scope (if defined). If the work-context is loaded into the agenda, the user is asked about their value.

2. **Receive.** Upon termination, the work-context may contain values of interest to the sender. If the sender is a cooperative procedure they are bound to procedure variables. The binding takes place before anything else is done in the procedure. If the work-context is sent by a user the return values are presented to this user.

At run-time, bindings of the work-context instance is stored under receiving user’s inst/directory. This file is updated with respect to editing text, selections, numbers. The work-context file also contains information about the origin of the work-context, the cooperative procedure (if any), and the activity (if any). The identity of this file can be communicated by an event button, and those that listen on this event may read the values by accessing the file. The file is removed when an answer button is pressed.
6.4.3 Cooperative Procedure editor

With the Cooperative Procedure editor we model the dynamic aspects of organizational processes. Essentially, we develop process programs called Cooperative Procedures which can be executed by the Process Weaver runtime system. The execution is denoted as enactment because cooperative procedures communicate closely with people by means of work-contexts. A cooperative procedure is considered as a process fragment because it can be executed independently of other cooperative procedures.

Petri-net

Cooperative procedures use Petri-nets as their metaphor. A Petri-net is a directed graph containing two types of nodes, places and transitions, and the links between them.

- Places. At each point in time, a subset of the net’s places will be filled with tokens, denoting the process state.

- Transitions. A transition has a set of input places and a set of output places. If all its input places contain a token and its condition is true, its action is performed and a token is created in all output places.

The semantics of different configurations is depicted in Figure 6.2. A transition can have any number of input and output places, and a single place can be used by many transitions. Parallel threads are created when a transition produces tokens in more than one output place. Two or more threads are synchronized when a transition needs tokens from more than one input place in order to “fire”.

Cooperative Procedure

In a Cooperative Procedure, transitions are split into a condition part and an action part. Furthermore, both conditions and actions may be one of several predefined types, these types are explained in two following sections. A condition acts as a guard, a transition cannot fires unless the condition is true. When the transition fire, the action part is executed.

A Cooperative Procedure communicates with users by sending Work Contexts to their Agendas. This is done in an asynchronous manner – according to “firing” transitions in the Petri-net. The typical usage is to send a work-context to the user in one transition, and then wait for an answer button pressed event in the subsequent transition. Meanwhile, this particular thread of control remains in the work-context. When the answer button pressed event occurs, and the work-context terminates, the thread of control is brought back to the waiting transition.

A cooperative procedure may invoke another procedure and then wait for its termination (synchronous), or start another procedure while it continues its execution (asynchronous). A procedure can be synchronized with another procedure by waiting for a state to be reached by this procedure. Two cooperative procedures may exchange data in three different ways:

148
1. **Input and output.** By binding input and output variables upon procedure invocation and termination.

2. **Messages.** By sending and receiving messages on the Broadcast Message Server.

3. **Files.** Read and write on files, i.e., by means of the Unix operating system.

At run-time a directory is created at the user’s inst/directory containing a complete textual description of the running procedure. This directory is removed when the procedure terminates. If not bound at this point the output is lost.

**The CoShell language**

The use of condition and action parts in Cooperative Procedures relies upon an understanding of the CoShell language. The CoShell language is the language provided by the Process Weaver environment. The language is interpreted at runtime and can be characterized as follows:

- Variables are untyped. The scope of a variable is local to a cooperative procedure. The scope is global with respect to all transitions within the procedure.
- Any variable can be declared as being an input and/or an output. Roles and documents are nothing more than variables containing references to users and files (by convention).
- A procedure may invoke another procedure, in this case the input/output binding of variables between the two procedures must be explicitly defined.
• Code is associated with transitions.
  
  – **Condition.** A logical expression over CoShell variables and functions. The expression is evaluated when there are tokens in all input places. The action part is executed if the expression evaluates to TRUE.
  
  – **Action.** The action is generally divided into two parts, a “first execute” part followed by an execution part specific to the action type (explained later). The “first execute” part is used for general calculations, function calls, variable binding, etc.

• WEAVER.lib includes many useful functions, e.g., list manipulating functions, file handling, invoking Unix commands, etc.

• Default CoShell variables give information about the context at run-time, e.g. the login name of the user that started the procedure.

A process programmer needs a detailed understanding of the CoShell language in order to make useful cooperative procedures. The Petri-net gives a rough indication of the process, but one needs to read the CoShell code to fully understand the process, e.g., to understand the selection among two alternative transitions.

### Condition Types

This section describes the possible condition types. Each condition type has an editor in which the process programmer enters CoShell statements in the available *information fields.*

• **Empty.** No condition, the action is executed.

• **Work-context condition.** Waiting for an answer from a work-context previously sent to a user. Information fields: Work-context ID, Answer button ID, and a logical expression. The condition is true if the Answer button ID of the Work-context ID is pressed, and the logical expression is true.

• **CoShell condition.** Logical expression in the CoShell language.

• **Procedure condition.** Waiting for a given state reached by another cooperative procedure previously launched by the procedure itself (a child procedure). Information fields: Procedure instance ID, and States (a list of places). The condition is true when the named sub-procedure has tokens in all the specified places. An empty list of places means that the sub-procedure must finish before the condition is true.

• **Event condition.** Waiting for an event on the global Broadcast Message Server (BMS). Information fields: Event specification and a logical expression. The condition is true if the specified event arrives and the logical expression is true. An event specification acts as a filter on BMS events according to a predefined message format (i.e. Event type, Tool class, Operation, Message ID, Context, and Data).
Collect condition. Indicate how to collect answers of a work-context sent previously to a group of people. This is similar to the Work-context condition, but in addition one must specify the Do and Do Until parts. The Do part is executed each time one of the Work-contexts is terminated, this is done until the Do Until part becomes true. The typical usage is to increment a counter in the Do part until the counter matches the number of work-contexts sent. A collect condition can also be specified for events, this is similar to the Event condition with analogous Do and Do Until parts.

The associated graphical symbols are depicted in Figure 6.3. Here, the empty condition is used in transition \( t_1 \) and \( t_4 \), the work-context condition in \( t_3 \), the CoShell condition in \( t_0 \), the procedure condition in \( t_6 \), the event condition in \( t_5 \), and the collect condition in \( t_2 \).

Figure 6.3: Cooperative Procedure: Transition symbols

151
**Action Types**

This section describes the possible action types. As for conditions, each type has an editor in which the process programmer enters CoShell statements in the available information fields.

- **Empty action.** No action. Used when we are only interested in the condition part of the transition.

- **Work-context action.** To send a work-context to a user. Information fields: First execute, Send to, and Work-context model. The “Send to” specifies the variable that should contain the user identity at runtime, the “Work-context model” must contain the name of the work-context to be sent.

- **Distributing action.** To send a work-context to a group of users. Information fields: First execute, For each in, Send to, For each execute, and Work-context model. The “For each in” field specifies a variable containing a list of user identities. The “Send to” field should contain a variable successively assigned to each list element, typically used in CoShell code specified in the “For each execute” part. In the “For each execute” part we may bind different information to each work-context, furthermore, we may register work-context identities such that the subsequent answers can be properly identified.

- **Procedure action.** To launch asynchronously a sub-procedure, which becomes a child of the cooperative procedure at run-time. The sub-procedure instance name is assigned to a variable for later use, e.g., in a condition. Information fields: First execute, Model name, and Inputs. “Model name” must contain the name of the cooperative procedure to be executed. “Inputs” should specify input values to the child procedure. This is done by listing the input variables of the child that we want to give initial values, and for each variable we assign a CoShell expression (executed in the context of the calling procedure). We may bind any variable, we are not restricted to those that are declared as inputs or outputs.

- **Synchronous Procedure action.** To launch synchronously a sub-procedure. When this action is taken, the execution of the current procedure is temporarily halted until the sub-procedure terminates (i.e., tokens in the parent output places are not produced until the child terminates, a child terminates when it produces a token in an output place). The information fields are similar to the procedure action, but there is an additional output binding part. The output binding is analogous to the input binding.

- **CoShell action.** Execute CoShell statements.

- **Event action.** To send an event on the global BMS. The editor is similar to the event condition editor, but without the logical expression part.

The associated graphical symbols are depicted in Figure 6.3. Here, the *work-context* action is used in t0, the *distributing* action in t1, the *procedure* action in t4, the *synchronous procedure* action in t2, the *CoShell* action in t3 and t5, and the *event* action in t6. The *empty* action is not shown.
6.5 Implementation

In this section we describe concerns in the implementation process. We describe differences in the two implementation environments, and consequences for the respective implementations. Furthermore, we describe the two processes from an implementation perspective, resulting in models which deviates from the ones presented in Chapter 4.

6.5.1 Implementation context and preparations

Both the Change management and the Document review processes were implemented in both Process Weaver and PCMS*CTS. The implementation was done as a student project during the autumn of 1995, and is documented in the EVEPS project report [45]. The EVEPS project was performed by six students within the context of a programming course at the Norwegian University of Science and Technology. This programming course is attended by fourth year students at the Department of Computer Systems. Students are forced into randomly selected groups and are then given a problem that is to be solved within three months, involving all phases in software development from requirements elicitation to the presentation of the final software product. Problem statements in these projects are sought from industry, or at least to be closely related to industry. Each project has an internal supervisor and an external customer. In the case of the EVEPS project, the problem statement was formulated by the thesis author in cooperation with the Company and SINTEF personnel\(^1\) working on the PERFECT project. The thesis author was the supervisor of this project, and the Company was the official customer. However, SINTEF personnel acted as customer in the weekly project meetings, on behalf of Company personnel.

The following practical preparations had been done by the thesis author before the student group started their work:

1. Process descriptions. Process descriptions of the change management and the document review processes had been made and was given to the students (as presented in Chapter 4).

2. PCMS installation. PCMS had been installed at the university, and a subset of the process owner's product structure had been imported. This was the part hierarchy depicted in Figure 3.8 (page 53), with a total of 676 PCMS items, including 102 change reports and 71 review reports.

3. Process Weaver training. Process Weaver had previously been installed at the university for other purposes. The thesis author was already familiar with the environment and had prepared example implementations and course material for student training.

Based on the process descriptions, and supplemented with questions to the process owners, the students had to learn how the two Company processes were performed manually. The process owners were available for questions during the whole project period. The students had to learn about PCMS*CTS and Process Weaver paradigms and possibilities, as both

\(^1\)Tor Stålhane and Svein Hallsteinsen.
these tools were unknown to them. Furthermore, the students should find reasonable process automation boundaries which had to be acceptable to the process owners. Discussions with Company personnel led to the following overall requirements:

1. There should be at least as much information present in the new change and review reports as in the present reports.
2. It was acceptable to replace printed change and review reports with equivalent electronic information.
3. It was not acceptable to replace printed documents in cases where documents should be read by others, e.g., for the purpose of document review or control.

Furthermore, from an experimental viewpoint it was a requirement that the implementations were comparable. This implies that one must view the implementations as related and not make completely different design decisions in the different implementations.

6.5.2 The two processes

In Chapter 4 we modeled the two processes for the purpose of process understanding. In this Chapter the modeling goal is to implement the processes in the two selected environments, and to use the implementation in the experiment reported in the next Chapter. In this context we wanted the two implementations to be comparable on the level of process steps, which caused the process decompositions to be slightly regrouped and renamed.

The two processes were divided into six process steps. The six process steps were maintained in both the manually performed process and in the two supported by process technology. In the experiment, the time used was recorded according to these six steps. As a consequence of this, a change process can be compared on the basis of individual process steps across its Manual PCMS, PCMS*CTS, and Process Weaver execution, and similarly for a review process. In the following we describe the process steps according to this new grouping.

Change management

This process is executed if previously approved documents must be updated, regardless of reason. For each CM process there is an associated Change Report (CR) where relevant information is documented (see page 96). When the process is finished, all affected documents will have new revision numbers and status APPROVED.

An overview of the CM process is depicted in Figure 6.4. The CM process definition is divided into six process steps. Process steps correspond to changing process roles, and the possibility to have different process performers at each process step. The six process steps are as follows:

1. Create change report. A change-reporter creates a change report and fills in the information known at this point in time, typically a description of the problem.
2. Appoint change responsible. The project’s change-manager writes the name of the person that is to be responsible for this particular change report.
3. *Find solution.* The change-responsible is responsible for finding a solution to the problem. He may approve the solution suggested by the change-reporter, he may find a solution himself, or he may call a meeting where the problem is discussed. When a satisfactory solution is found, he extracts the change report from PCMS and describes the solution. He ensures that all documents affected are listed in the change report with correct revision number, and that every document is allocated a document editor. The change report status field is set to SOLUTION-APPROVED.

4. *Perform changes.* Each editor receives a paper copy of the change report. On the basis of information found in the change report, they extract referenced documents from PCMS. The editors do the necessary changes until the change-responsible is satisfied. The editors then return their documents to PCMS and set document status to DRAFT.

5. *Finish change report.* When all editors have changed their documents, and every document has obtained PCMS status DRAFT, the change-responsible edits the change report status field from SOLUTION-APPROVED to DONE. He also changes the PCMS status from UNDER-WORK to DRAFT.

6. *Approve documents.* When the change report has status DONE, the change-manager approves each document, and then finally approves the change report.

If we compare this process description with the one of Section 4.7 (page 96), we see that it is similar to the top level activity diagram depicted in Figure 4.25 (page 99). There are two main differences: First, the “initiate change” process step has been expanded into its components, namely “create change report” and “appoint change responsible”. Second, the “suspend changes” process step has been incorporated into the “find solution” process step. The other process steps in the two descriptions correspond to each other.

**Document review**

This process is executed when one or more authors decide to submit their DRAFT documents to a formal review. Each document must be reviewed at least once to become approved. For each DR process there is an associated Review Report (RR) where relevant information is documented (see page 89). When the process ends, new document revisions may have been created, and the document is either in state REJECTED or DRAFT (waiting for project manager approval).
An overview of the DR process is depicted in Figure 6.5. As for the CM process definition, the DR process definition is divided into six process steps according to changing process roles. In the DR process, different documents may take different paths in the process graph. The path depends upon the document status after the review meeting, this is illustrated in Figure 6.5 by annotating the arrows with status values (NOK, PARTIAL, FIX, and OK). The six process steps are as follows:

1. **Create review report.** The author creates a review report and fills in the name of the review manager, review secretary, and reviewers, describes the scope of the review, and notes the time and place for the review meeting. He ensures that every document is set to DRAFT, except the review report that maintains its UNDER-WORK status. He then distributes the review report and the associated documents to all reviewers.

2. **Update review report.** After the review, the results are documented in the review report by the review-secretary. Each document receives a review result that is either OK, FIX, PARTIAL, or NOK. The PCMS status is unchanged.

3. **Change document status.** The meeting-manager changes the status of documents with review result different from OK, from DRAFT to REJECTED. He then distributes the documents to their respective authors. OK documents need no further attention. FIX documents need to be corrected and controlled. PARTIAL and NOK documents need a new review process and become irrelevant to the current one.

4. **Correct and control documents.** Only for FIX documents. Authors edit their documents according to the review findings and they are possibly controlled by an appointed controller. New revisions of the documents are produced. PCMS status is set to DRAFT.

5. **Finish review report.** The meeting manager collects updated documents from the authors. When all OK and FIX documents have been received, the meeting manager updates the review report and sets its PCMS status to DRAFT.

6. **Approve review report.** The review report is set to APPROVED by the project manager. This is the exit criteria for the overall review process. Documents are not approved by the review process; this is done by the project manager in some subsequent approval process.

If we compare this process description with the one of Section 4.6 (page 89), we see that it is almost identical to the top level activity diagram depicted in Figure 4.22. The process steps
are maintained, but a few have changed names, e.g., “review meeting” is changed to “update review report”. Furthermore, we see that non-automatable process steps are ignored in the new process description, e.g., “prepare meeting” and “meeting” in Figure 4.23 (page 94). This reflects a change of focus, as now, the primary interest is to describe the interaction between the computer system and process agents.

6.5.3 The three execution environments

This section gives an overview of the three execution environments, namely Manual PCMS, PCMS*CTS, and Process Weaver. Each execution environment is presented according to the following three perspectives:

2. Work environment. How process agents interact with the supporting environment.
3. Review and Change reports. How review and change reports are created, updated, and presented to process agents.

In the following we will describe the three environments from these perspectives. The purpose of the three perspectives is to facilitate comparisons between the environments, identifying similarities and differences.

Manual PCMS

Since data on the administrative overhead was not available for the manually performed processes, we had to measure it in the same way as we measured the overhead for the automated executions. This was done as part of the experiment, denoted as Manual PCMS.

With Manual PCMS the two processes are not supported by means of process technology. This implies that process chaining and notification are performed manually by the involved process agents. The work environment offered to process agents is the PCMS “product browser” and a text editor\(^2\). The RR and CR reports are implemented as ordinary PCMS items, extracted and returned by means of the product browser, and updated by means of the text editor.

Process support.

In Manual PCMS the two process types are performed as described in the Engineering Handbook. Process instances are supported by their respective reports as each report describes agent–role allocations, affected documents, agent–document allocations, status information, etc. Process chaining and notification are done manually, process agents are notifying each other, using the printed reports as tokens.

Work environment.

With Manual PCMS, the work environment offered to process agents is the standard PCMS “product browser”, identical to the current work environment in the Company. The PCMS

\(^2\)The Unix operating system is implicit in all described environments.
“product browser” allows the process agent to navigate in the PCMS part structure, select a part, and view and select item revisions. A selected item can be extracted to the agent’s current Unix directory. In the Unix directory, the item is a normal file and can be loaded into an editor according to file format, in our case all files were plain text files. When the agent has made his updates he may return the file to PCMS which creates a new item revision (if the extracted file was not in its initial lifecycle state). The agent may also use the product browser to action items. An item is actioned by selecting the item and choosing “action item” in the browser menu. The alternative “next states” are displayed and the agent can select among these.

![Diagram showing lifecycle states](image)

**Figure 6.6: The RR and CR lifecycle**

*Review and Change reports.*

In Manual PCMS both review and change reports were implemented as ordinary PCMS items, identical to the reports in the studied company. The original DOC-1 lifecycle, depicted in Figure 6.6, was applied to both reports. Both the original states and roles were left unchanged with respect to Company definitions. All process agents were allocated to the *author* role, while only one agent was allocated to the *reviewer* role, namely the project manager. The process agents create the reports by running two scripts called CCR <number> for change reports and CRR <number> for review reports. The “number” must be unique, typically an increment of the previous report number. These scripts create the reports within PCMS in their initial lifecycle state (UNDER-WORK). All process agents may extract, update, and return these reports, only restricted by the PCMS concurrency control.

**PCMS*CTS**

With PCMS*CTS the two processes are supported by PCMS*CTS Change Documents. Through the use of PCMS*CTS Change Documents the two processes are partially supported with respect to process chaining and notification. The *work environment* offered to process agents is the Change Document itself, supplemented with the use of the PCMS “product browser”. The *RR and CR reports* are fully supported through their Change Document implementation.

PCMS*CTS does not fit perfectly into the three perspectives, because the Change Document merges these perspectives. The change document is, all at once, the process support, the work environment, and the developed report.

*Process support.*

With PCMS*CTS, the two processes are partially supported with respect to process chaining
and notification. Partially, because only agents responsible for updating the change documents are supported, agents responsible for updating the affected documents are not supported:

1. **Change documents (CD).** Agents responsible for updating CD receive notifications according to state transitions in the CD's lifecycle. When a CD is actioned to a new state, the agents connected to all its out-transitions are notified by electronic-mail, furthermore, the CD enters the pending list of these agents.

2. **Affected documents.** In the two PCMS*CTS implementents, agents responsible for updating the affected documents are not part of the CD lifecycle, and will thus not receive the CD in their pending list. Notifications to agents who should perform document updates are made as in the Manual process, namely as printed RR and CR reports.

It is possible to include the roles of “document editors” in the RR and CR lifecycles, and not doing it is a process design decision. The main reason for this decision is that none of the editors should be allowed to update the CD, only to read it. It is then better to provide them with a paper copy which is more readable and cannot be “misused”.

![Diagram of the Review Report lifecycle](image)

**Figure 6.7: The Review Report lifecycle**

The PCMS lifecycle associated to the *Review Report* is depicted in Figure 6.7. The lifecycle is extended with two states with respect to the normal DOC-1 lifecycle. In the ANALYSIS phase the process agent may relate documents as “affected” but not extract documents, causing the “in-response-to” relationship to be created. In the WORK phase the process agent can extract, update, and return “affected” documents, but not relate new documents as “affected”. In the FROZEN phase no changes to the relationships or related items are permitted, the process agent can only action the RR and update RR attributes. In the CLOSED phase no change is permitted.

The identity of the agent who creates the RR will automatically be stored in the predefined “originator” attribute, always present in PCMS change documents. This is utilized such that the *author* role becomes *originator* within the process instance. The *project manager* role is played by the same agent in all review process instances, this role is therefore declared as a PCMS static role. The *secretary* and *meeting manager* roles are played by different agents in different review process instances, these roles are therefore declared as PCMS dynamic roles. Note that the *author* is only mentioned in the lifecycle as “originator” and not as document “editor”. The work of updating the documents after the review meeting is reflected in the lifecycle by the WORK and FINISH-DOC states, but *authors* will not receive the RR in their pending list when they are to update FIX documents.

---

3PCMS dynamic and static roles: See page 141.
The PCMS lifecycle associated to the Change Report is depicted in Figure 6.8. With respect to phases the only difference between the CR and RR lifecycles is the CREATE phase. The CREATE phase is almost identical to the ANALYSIS phase, with the difference that lower level change documents cannot be related to the change document while it is in the CREATE phase. With respect to states, this lifecycles has a WAIT state to which the CR is actioned if the CR is decided to be suspended/postponed. At the end of the ANALYSIS phase the change responsible may action the CR to WAIT, meaning that the change is going to be implemented, but not now. When the CR is to be implemented it is actioned to ANALYZE by the change manager, and the process continues as before.

As for RRs, the identity of the agent who creates the CR will automatically be stored in the predefined “originator” attribute. The change manager role is played by the same agent in all review process instances, this role is therefore declared as a PCMS static role. The change responsible is played by different agents in different review process instances, and is therefore declared as a PCMS dynamic role. Note that the editors are not mentioned in the CR lifecycle. The work of changing the documents is reflected in the CR lifecycle by the WORK state, but editors will never receive the CR in their pending lists.

Work environment.

The work environment offered to the process agent is the PCMS*CTS Change Document. The CD is tailored to the process through the implementation of CD types, in our case the CHANGE and REVIEW types. For each CD type one can specify valid attributes. Attributes are referenced in the CD template, in which the CD layout is defined.

The interface to a Change Document is through CD attributes. CD attributes are categorized into PCMS defined and User defined attributes. These attributes are written in the CD template in a user defined layout position. At runtime these are interpreted by PCMS*CTS and substituted with appropriate values or input fields. The most important PCMS defined attributes and their use are as follows:

1. %description%: This attribute is substituted with a general text area, this text area can be updated by a normal text editor. It is used to describe the problem and the solution in the change report, and to describe the scope in the review report.

2. %user_roles%: This attribute is substituted with role-agent allocations. Static roles are only displayed, dynamic roles can be updated by means of the PCMS*CTS “delegate”
dialogue.

3. `%affected_item%`: This attribute is substituted with a list of all items currently related to the CD. Each entry includes information such as the item-id, the type of relation, the time of the relation, and the user-id of the agent that made the relation. Affected items are updated by means of the PCMS*CTS “relate item” dialogue.

The *User defined attributes* can be categorized into single valued and multi valued attributes.

1. Single valued attributes. The attribute is substituted with an input field in the CD. Each single attribute must have a data type, e.g. characters, numbers, and enumerated. Single-valued attributes are used to store process information such as estimated and actual effort.

2. Multi valued attributes. The attribute is substituted with multi-row, multi-column input fields in the CD. The programmer may compose his own multi-valued attributes by using existing single-valued attributes, one single-valued attribute per column. PCMS*CTS provides a spreadsheet like interface to multi-valued attributes. Multi-valued attributes are used to store process information such as the annotations in the review report, and the enumerated problems in the change report.

A range of properties can be associated to attributes, e.g. updatable, mandatory, and restricted to users and roles. Attributes can also be grouped according to role sections, and visibility controlled according to these sections. However, in our case all attributes were defined as updatable, not mandatory, and not restricted in any way.

A CD is created by choosing “Create/Change Document” in the PCMS*CTS menu. PCMS*CTS prompts the *originator* for the PCMS product and CD type. The *originator* chooses among the two implemented types, CHANGE or REVIEW, creating the corresponding CD in its HELD phase. The CD may then prompt the *originator* for some attribute values, depending on the CD implementation. The originator may update the attributes and then save and close the CD. This causes the CD to enter its initial lifecycle state in the CREATE or ANALYSIS phase. The CD then enters the pending list of the agents allocated to the role defining the first state transition in the CD lifecycle.

In general, any agent with a CD in his pending list can update document attributes, only restricted by the phase concept, e.g., he cannot add new documents in the work phase. These agents can also action the CD to its next lifecycle state. In cases where the CD is in the pending list of several agents, they must be aware that other agents can update the CD. A CD in the pending list can be *opened* and *closed*, when opened the current attribute values are shown. However, when it is *open* it is not automatically updated when other agents updates the document. It is possible to define one of the agents as the leader, and in this case only the leader can update the CD attributes.

The *affected documents* (DOCs) can be extracted, updated, and returned, by means of the PCMS “product browser”. This way it becomes identical to the manually performed process described in the previous section. However, since the affected documents can be found by means of the CD’s “affected” relationship, this work can be eliminated. PCMS*CTS does not offer such functionality, but it can be achieved through the use of the PCMS/API. This

161
was done in connection with the PCMS*CTS implementation of the two processes. Here, the following programs were developed to support the review *authors* and change *editors* with operations on sets of affected documents.

1. ToDraft <RR-number>. This program actions all affected DOCs from UNDER-WORK to DRAFT.

2. RejectDocs <RR-number>. This program actions all affected DOCs with review result equal to NOK, PARTIAL, or FIX, from DRAFT to REJECTED.

3. Redo <RR-number>. This program extracts DOCs with review result equal to FIX from the PCMS repository to the current user directory.

4. RedoReturn <RR-number>. This program returns all DOCs extracted by the Redo program. New revisions of DOCs are created, and all DOCs are actioned from UNDER-WORK to DRAFT.

5. Extract <CR-number> <agent>. This program extracts all the affected DOCs with the value *agent* in the DOC's *editor* attribute.

6. Return <CR-number> <editor>. This program returns all DOCs extracted by the Extract program. New revisions of DOCs are created, and all DOCs are actioned from UNDER-WORK to DRAFT.

7. Approve <CR-number>. This program actions all the affected DOCs from DRAFT to APPROVED.

All these programs utilize the CD's "affected" relationships. Some of the programs utilize attributes of affected DOCs. The programs which return DOCs to PCMS create the CD's "in-response-to" relationships.

Similar programs\(^4\) are applied in the Process Weaver implementation, but in this case they are performed automatically. Since this is not an embedded advantage of Process Weaver, we felt that these scripts had also to be used in the PCMS*CTS case. Note that these scripts cannot be used in the Manual case, because in this case the list of "affected documents" only exists as a free text description in the RR and CR reports.

---

\(^4\)In this case the list of "affected" documents is implemented as a program variable, storing the names of affected items.
Both the *Review report* and the *Change report* were implemented as PCMS*CTS Change Documents. The layout of the RR and CR reports are almost identical to their manual counterparts. The main difference is that the CD not only stores the current attribute values, but also the complete value history. The two reports included the two predefined PCMS attributes called:

1. **%action_history%**: Lists one entry for each time the report has been actioned.
2. **%update_history%**: Lists the complete history of attributes, i.e., one entry for each attribute, each time its value has changed.

Furthermore, the Change Documents can be retrieved with the attribute values at any previous action point. This ensures that process agents can inspect historical states of individual change and review reports. This cannot be achieved in either of the other two implementations.

PCMS*CTS offers a wide range of reports, divided into *pending* and *catalogue* reports. The pending reports cover change documents which are currently active in the system, and catalogue reports cover all change documents in the system. This ensures that process agents can get an overview of current and historical change and review reports. This cannot be achieved in either of the other two implementations.

Finally, PCMS*CTS creates relationships in the database between change documents and the affected documents. This can be utilized to create customized reports and programs that analyze and present historical data about change and review processes. This cannot be achieved in either of the other two implementations.

### Process Weaver

With Process Weaver the two processes are supported by Cooperative Procedures. This implies that they are fully supported with respect to *process chaining and notification*. The *work environment* offered to process agents is the Process Weaver *Agenda* and a set of developed *Work Contexts*. The **RR and CR reports** are controlled by the developed cooperative procedures during process execution. This is done to prohibit a direct manipulation by process agents, however, the final CR and RR reports are unchanged with respect to the manually performed process.

**Process support.**

With Process Weaver the two processes are implemented as Cooperative Procedures. A process step is implemented as one or more Work-Contexts sent from the cooperative procedure to one or more process agents. The *chaining* among process steps is fully and freely described by the process programmer. A work-context arriving in an agent's agenda is a *notification* to the agent about a process event. When the process agent opens the work-context, he sees the work-environment for this particular process step. He may perform work in a work-context until he presses an “answer” button, when this happens the work-context disappears from the agenda, and the control is transferred to the sending cooperative procedure.
The CoShell code associated to cooperative procedure transitions is capable of executing Unix scripts and programs. PCMS provides an Application Programmers Interface (API), a C-interface to PCMS functions. These two features are utilized in the change and review process programs. A set of C-programs were developed, these programs are called with input parameters from the cooperative procedure. The programs execute PCMS functions and return results to the calling cooperative procedure. In the developed process programs this was utilized to:

1. **Extract items.** In process steps where process agents are to update PCMS items the process program extracts the relevant items from PCMS into a Unix directory. Relevant items are either the reports or documents mentioned in the reports. The extracted items are stored in the Unix directory before the corresponding work-context is sent to the agent’s agenda.

2. **Load files into editors.** The work-contexts contains “document symbols” for documents relevant to the corresponding process step. When the process agent opens a “document symbol” an editor is started on the corresponding document file. He then edits and saves the document by means of normal editor functionality.

3. **Return items.** When the process agent presses the answer button the control is transferred back to the process program. The process program then performs a PCMS return on all extracted files, placing the files under PCMS control.

4. **Action items.** The process program actions PCMS items to new lifecycle states based on the programmed process knowledge. This is normally performed after the process agent has pressed an answer button in a work-context, confirming that the process has reached a point where the items can be actioned to their next lifecycle states.

The consequence is that with the Process Weaver implementation, process agents never use the PCMS “product browser” to extract, return, or action items. Furthermore, the workspace becomes “invisible” to process agents, as they only see editors containing the files to be edited.

**Work environment.**

The work environment offered to process agents in Process Weaver is the Work Context (WC). The process programmer applies the work context editor to make work environments tailored to the corresponding process steps. The possibilities of the work context editor is described in section 6.4.2, page 145. The following work context objects were applied in work-contexts associated to the two process programs:

1. **Answer button.** To allow agents to finish the process step.

2. **Selection list.** To allocate agents to roles, and to select documents to be controlled.

3. **Edit list.** To allow agents to write and update the list of affected documents, to allocate editors to documents, and to update the status of individual documents.

4. **Text area.** To display messages and work-context descriptions to the performing agents.

5. **Editable text area.** To allow agents to write and update report descriptions and input fields, e.g. problem description and revealing activity.
6. *Numbers*. To allow agents to write and update estimated and actual effort.

7. *Documents*. To allow agents to read and update process documents.

At runtime, these work-context objects are given actual values by the sending cooperative procedure. A work-context instance is received by a process agent in his agenda. While the work-context instance is in the agenda, it is “owned” by the receiving agent. The process agent enters information in updatable work-context objects, this information is stored in a local file until the agent presses an answer button. At this point the data in the local file is bound to variables in the sending cooperative procedure.

*Review and Change reports.*

With respect to PCMS, both review and change reports were maintained as ordinary PCMS items, identical to the *Manual PCMS* case. However, with the Process Weaver implementation these reports were extracted or returned by the process programs, not by process agents.

Furthermore, the review and change reports represent a problem in the Process Weaver implementation for the following reasons: First, it was a requirement that these reports should exist and contain the same information as in the manually performed process. Second, these reports contain information needed by the two process programs, e.g., agent–role allocations and affected documents, thus, the information must be present in cooperative procedure variables. Third, the information found in reports and in cooperative procedure variables should be consistent.

The developed solution is as follows: First, the process agents were not allowed to edit the reports in a direct manner, i.e., data could only be entered by means of work-context input fields (non document objects). Second, the RR and CR reports were presented to the process agents as Process Weaver “read-only documents”. The presented reports were generated by the two process programs on the basis of data entered in work-context input fields. Updateable text areas were stored in separate files managed by the process programs. When changed, these files were automatically concatenated and presented as an updated report.

**6.6 The six alternatives**

This section describes the *Document review* and *Change management* processes as they were performed in the enactment experiment. Each of the two processes is described three times, one for each of the three execution alternatives which are *MANUAL, PCMS*<sup>*</sup>*CTS*, and *Process Weaver*.

**6.6.1 MANUAL – Document review**

Performed *Manually*, the *Document review* process works as follows:

1. *Create RR*. The author creates the RR as an ordinary PCMS item by running a PCMS script called CRR `<number>`. The RR number must be unique, i.e., an increment by one from the previous RR number. This script creates the RR from the RR template.
The RR now exists as an PCMS item in its initial lifecycle state (UNDER-WORK). The author must:

(a) Extract the RR from PCMS.
(b) Load the RR into his editor.
(c) Fill in the following fields: Participants, meeting manager, secretary, time, place, and scope.
(d) List documents to be reviewed (DOCs). For each DOC, the full DOC item name is keyed into the RR.
(e) Action all DOC items from UNDER-WORK to DRAFT. The author must write the RR identifier in the action item comment field. The comment is prompted for by PCMS each time a user performs an action item command.
(f) Print the RR.
(g) Return the RR to PCMS.
(h) Copy the RR, write the names of participants on the paper copies, and put copies in the out-box.

2. Update RR. After the review meeting, the review results are documented in the RR by the secretary. The secretary must:

(a) Extract the RR from PCMS.
(b) Load the RR into his editor.
(c) Update review result for each DOC. The review result can be one of OK, FIX, PARTIAL, or NOK).
(d) Update controller for each DOC with review result equal to FIX.
(e) Update actual effort fields.
(f) Update annotation fields.
(g) Print the RR, give it to the meeting manager.
(h) Return the RR to PCMS.

3. Change document status. The DOCs are now partitioned into four sets according to the review result. The meeting manager must:

(a) Action all DOC items with review result equal to NOK, FIX, or PARTIAL, from DRAFT to REJECTED.
(b) Distribute the CR and FIX DOCs to their respective authors.

4. Correct and control documents. For each FIX DOC, the author must:

(a) Extract the DOC from PCMS.
(b) Load the DOC into his editor.
(c) Update the DOC according to RR annotations.
(d) Return the DOC to PCMS.
(e) Action the DOC from UNDER-WORK to DRAFT.
5. **Finish RR.** The *meeting manager* collects updated documents from the author(s). When all F1X documents have been received, the *meeting manager* must:

   (a) Extract the RR from PCMS.
   (b) Load the RR into his editor.
   (c) Update the RR *review result* of all F1X DOCs from F1X to OK.
   (d) Return the RR to PCMS.
   (e) Action the RR from UNDER-WORK to DRAFT.

6. **Approve RR.** The RR is to be APPROVED by the project manager. This is the exit criterion for the overall review process. Documents are not approved by the review process; this is done by the project manager in some subsequent approval process.

   (a) Action the RR from DRAFT to APPROVED.

### 6.6.2 PCMS*CTS – Document review

The *Document review* process, automated by means of *PCMS* *CTS* works as follows:

1. **Create RR.** The *author* chooses a PCMS*CTS Change Document of type REVIEW, creating an RR in its HELD phase. The RR then prompts the *author* for some attribute values. The *author* must:

   (a) Fill in meeting *participants* by selecting these from a candidate list.
   (b) Fill in values for the RR meeting attributes (place, time, date).
   (c) Describe the scope of the review in the PCMS “description” attribute.
   (d) Save and close. When the *author* is satisfied he saves and closes the change report which enters its initial lifecycle state (START) in the ANALYSIS phase.
   (e) Select and open RR. The RR enters the pending list of the agent playing the *author* role, this because the *author* is the initial role in the RR lifecycle. The *author* selects and opens the RR from his pending list.
   (f) Select and relate items. The *author* selects “Relate items” from a RR menu. He may specify a part/item filter, and then select “search”. The search results in a list of items (DOCs). He selects items from this list. The selected items are related as affected. He may also unrelate affected items in case of mistakes.
   (g) Delegate roles. The *author* is responsible for allocating agents to two roles, namely the *meeting manager* and the *secretary*. This is done by means of the PCMS*CTS delegate dialogue. Here, he selects a role and then selects one of the agents in the provided candidate list.
   (h) Action RR. When the *author* is satisfied he action the RR to its next lifecycle state which is WORK (WORK phase).
   (i) Close the RR.
   (j) Action DOCs. The *author* must actions all the affected documents from UNDER-WORK to DRAFT. This is done by executing the ToDraft <RR-number> program.
2. **Update RR.** The RR enters the *secretary's pending list. The* *secretary must:

(a) Update DOC attributes. For each affected document, the *secretary must update the *review result and controller attributes. This cannot be done from the RR, because PCMS*CTS does not support attribute update from the list of affected documents. The consequence is that each document must be found manually by using the "Product browser". When found, the document is selected and the attributes updated by using "Edit attributes" in the browser menu:

- Update DOC review result. Write one of the four possible values (OK, FIX, PARTIAL, NOK) in the review result attribute.
- Update DOC controller. Write the user-id in the controller attribute.

(b) Select and open the RR from his pending list.

(c) Write annotations. Annotations are implemented as a multi-valued PCMS attribute. There is one row for each annotation, and one column for each of the annotation components (annotation number, place, responsible, annotation). The interface to a multi-valued attribute is a table with scrollbars in both dimensions, a single cell can be updated by selection and editing.

(d) Update actual effort. The actual time used on the review process, except the control of document updates.

(e) Action RR. When the *secretary is satisfied he action the RR to its next lifecycle state which is FINISH-DOC (WORK phase).

(f) Close the RR.

3. **Change document status.** The RR enters the *meeting manager's pending list. The *meeting manager must:

(a) Select and open the RR from his pending list.

(b) Action DOCs. All documents with review result equal to NOK, PARTIAL, or FIX, must be actioned from DRAFT to REJECTED. This is done by executing the RejectDocs <RR-number> program.

4. **Correct and control documents.** The *meeting manager is responsible for the RR while the documents are being updated and controlled.

(a) Extract DOCs. The *author extracts all documents with *review result equal to FIX. This is done by executing the Redo <RR-number> program.

(b) Edit DOCs. All FIX documents are now in the *author's workspace. He must start an editor with the different documents and make changes according to the RR, which he may browse or print.

(c) Control DOCs. The document control is performed in an informal way, the controller does not update the RR or the DOCs. The responsible agent can be found in the DOC's controller attribute (normally it is the meeting manager).

(d) Return DOCs. The *author returns all DOCs to PCMS, and furthermore, actions all DOCs from UNDER-WORK to DRAFT. This is done by executing the RedoReturn <RR-number> program.

168
5. **Finish RR.** The meeting manager collects updated documents from the author(s). When all FIX documents have been received, the meeting manager must:

(a) For each DOC, update the review status attribute. Each DOC must be found manually by using the PCMS “Product browser”. When found, the document is selected and the attribute updated by using “Edit attributes” in the browser menu. The project manager then updates the review status attribute from FIX to OK.

(b) Action RR. Action the RR to its next lifecycle state which is FINISH-RR (FROZEN phase).

(c) Close RR.

6. **Approve review report.** The RR enters the project manager’s pending list. The project manager must:

(a) Select and open the RR from his pending list.

(b) Action RR. Action the RR to its final lifecycle state which is APPROVED (CLOSED phase).

(c) Close RR.

### 6.6.3 Process Weaver – Document review

The Document review process, automated by means of Process Weaver works as follows:

1. **Create RR.** The author starts the PW cooperative procedure called review.cp.

   (a) The procedure prompts the author for three input values, the PCMS database-id, the PCMS product-id, and the Workspace. The Workspace must be a directory owned by the weaver user, files extracted from PCMS are stored in this directory while they are updated.

   (b) The author receives a work-context (WC) on his agenda. The author must:

   i. Open the WC.

   ii. Select process participants from a list of all project members (PW selection list).

   iii. Select exactly one meeting manager, and exactly one secretary from a PW “selection list”.

   iv. Write time and place for the review meeting in a PW “editable text area”.

   v. Write the scope of the review in a PW “editable text area”.

   vi. Write time used on review preparations in a PW “float number”.

   vii. Write the documents to be reviewed in a PW “edit list”. The author may here specify the item revisions to be reviewed. A complete list of all product documents in the state UNDER-WORK can be displayed from a PW read-only document. This list is generated by executing a PCMS query from the cooperative procedure’s CoShell code. Elements from this list can be copied into the “edit-list”.

   viii. Press the WC “done” button.
(c) Each participant receives a “Call for review meeting” work-context in their agenda. Each participant must:
   i. Open the WC.
   ii. Read information about time and place, involved roles and allocated agents, documents to be reviewed, and scope of the review. It is assumed that the actual DOCs are distributed as paper copies.
   iii. Write the time used on reading and annotating the DOCs.
   iv. Press the WC “done” button.

2. Update RR. The secretary receives a work-context in his agenda. This work-context is to be filled in after the review meeting has taken place. The secretary then updates the work-context according to the review protocol, made in the review meeting. The secretary must:
   (a) Open the WC.
   (b) Write the review status for each DOC. This is updated in the PW “edit list” which displays the DOC names. Each list element has a defined format which starts with the DOC name, and is followed by review status and controller. The latter two might be empty. The review result can be copied from a list of possible values, namely OK, FIX, PARTIAL, or NOK. The CoShell code updates the DOC review result attribute within PCMS.
   (c) Write the controller for each FIX DOC. This is also updated in the PW “edit list” which displays the DOC names. The value can be copied from a list of possible controllers. The CoShell code updates the DOC controller attribute within PCMS.
   (d) Write annotations. A list of enumerated annotations is made by means of a PW read-write document. The annotations are merged with the RR.
   (e) Write the actual effort. The actual effort is composed of time used on the review protocol, the duration of the meeting, and number of participants.
   (f) Press the WC “done” button.

3. Change document status. This process step is fully automated. All documents with review result equal to NOK, PARTIAL, or FIX, are actioned from DRAFT to REJECTED.

4. Correct and control documents. The author receives one WC for each of the reviewed DOCs having review status FIX, PARTIAL, or NOK. The WCs are different according to the review status. For each DOC work-context:
   (a) Open the WC.
   (b) PARTIAL or NOK: In these cases the author receives information about the review results, but the cooperative procedure does not wait for answers, since these documents need a new review process.
   (c) FIX: The author updates the FIX DOCs. Note that the control is not supported by a separate work-context. When the author has finished his updates, he prints the documents and distributes them to controllers. The controller reads and comments on the paper copy, and gives feedback to the author.
(d) Update the actual effort spent on control.
(e) Press the WC “done” button.

5. Finish RR. When the author has pressed the “done” button in all WC containing FIX
   DOCs, the RR is automatically actioned from UNDER-WORK to DRAFT.

6. Approve RR. The project manager receives a work-context in his agenda. The project
   manager must:

(a) Open the WC.
(b) Update the review status entry for each FIX DOC in the RR from FIX to OK.
(c) Press the WC “done” button. PCMS update: The review status of all FIX DOCs
   are updated by the CoShell code, furthermore, the RR is actioned from DRAFT
   to APPROVED.

6.6.4 MANUAL – Change management

Performed Manually, the Change management process works as follows:

1. Create CR. The change reporter creates the CR as an ordinary PCMS item by running
   a PCMS script called CCR <number>. The CR number must be unique, i.e., an
   increment by one from the previous CR number. This script creates the CR from a
   predefined CR template. The CR now exists as an PCMS item in its initial lifecycle
   state (UNDER-WORK). The change reporter must:

(a) Extract the CR from PCMS.
(b) Load the CR into his editor.
(c) Write the date, name of the change reporter, problem description, estimated effort,
   and revealing activity.
(d) Write UNDER-WORK in the CR change status field.
(e) Write other information known at that time, e.g., solution and affected documents.
(f) Print the CR and give it to the change manager, e.g., put it in his in-box.
(g) Return the CR to PCMS.

2. Appoint change responsible. The change manager writes the name of the change respon-
   sible on the CR paper copy, he then gives the CR to the appointed change responsible
   (e.g., his in-box).

3. Find solution. The change responsible reads the CR paper copy. When he has found a
   satisfactory solution he must:

(a) Extract the CR from PCMS.
(b) Load the CR into his editor.
(c) Update change responsible field.
(d) Update the problem description and the solution.
(e) Update the list of affected DOCs. The full name of each affected item is keyed into the CR.

(f) Allocate an editor to each DOC.

(g) Update estimated effort, problem number, introducing activity, and category.

(h) Update the CR change status to SOLUTION-APPROVED.

(i) Print the CR.

(j) Return the CR to PCMS.

(k) Distribute the CR paper copy to all editors.

4. Perform changes. Each editor receives a paper copy of the CR. For each DOC of which he is the editor he must:

(a) Extract the DOC from PCMS.

(b) Load the DOC into his editor.

(c) Update the DOC.

(d) Return the DOC to PCMS. A new item revision is created upon return, the item is in its initial lifecycle state (UNDER-WORK).

(e) Action the DOC from UNDER-WORK to DRAFT. The editor must write the CR identifier in the action item comment field (PCMS prompts for a comment to all action item operations).

5. Finish change report. When all editors have updated and actioned their documents to DRAFT, the change responsible must:

(a) Extract the CR from PCMS.

(b) Load the CR into his editor.

(c) Edit the CR change status field from SOLUTION-APPROVED to DONE.

(d) Return the CR to PCMS.

(e) Action the CR from UNDER-WORK to DRAFT.

6. Approve documents. When the CR change status field has the DONE value, the change manager must:

(a) Action each updated DOC from DRAFT to APPROVED.

(b) Action the CR from DRAFT to APPROVED.

6.6.5 PCMS*CTS – Change management

The Change management process, automated by means of PCMS*CTS works as follows:

1. Create change report. The change reporter chooses a PCMS*CTS Change Document of type CHANGE, creating a CR in its HELD phase. The CR then prompts the change reporter for some attribute values. The change reporter must:
(a) Update the revealing activity and estimated effort attributes.
(b) Save and close CR. This causes the CR to enter its initial lifecycle state (START) in the CREATE phase. The CR enters the pending list of the agent playing the change reporter role, this because the change reporter is the initial role in the CR lifecycle.
(c) Select and open the CR from the pending list.
(d) Write the problem description. He may also propose a solution and relate documents (described in “find solution” below).
(e) Action CR. Action the CR from START (CREATE phase) to CHECK-CR (CREATE phase).
(f) Close the CR.

2. Appoint change responsible. The CR enters the change manager’s pending list. Change manager is a static role, the role/agent is defined within the product structure, and therefore this agent is automatically bound to the change manager role within the Change Report. The change manager must:

(a) Select and open the CR from his pending list.
(b) Allocate change responsible. Allocate an agent to the change responsible dynamic role. This is done by means of the PCMS*CTS delegate dialogue. He must select the change responsible role and then choose one of the agents in the candidate list.
(c) Action CR. Action the CR from CHECK-CR (CREATE phase) to ANALYZE (ANALYSIS phase).
(d) Close the CR.

3. Find solution. The CR enters the change responsible’s pending list. The change responsible must:

(a) Select and open the CR from his pending list.
(b) Relate DOCs as affected. Select “relate-items” from a CR menu. He may then specify a part/item filter, and select “search”. The search creates a list of items, he selects items from this list. The selected item is related as affected. He may also unrelate affected items.
(c) Allocate editors. For each affected DOC, he must allocate an editor. This is done by updating the editor attribute value for each affected document. This cannot be done from the CR, i.e., PCMS*CTS does not support this, and each document must be found manually by using the “Product browser”. When found, the document is selected and the attribute updated by means of the edit-attributes command. The change responsible then writes the user-id of the responsible agent in the editor attribute.
(d) Update estimated effort, problem number, introducing activity, and category.
(e) Update the problem description and the solution. A PCMS Change Report only provides one general editing attribute, namely the “description”. Thus, both the problem description and the solution were described in this attribute.
(f) Print the CR. Distribute CR paper copies to all editors.
(g) Action CR. Action the CR from ANALYZE (ANALYSIS phase) to WORK (WORK phase).

(h) Close the CR.

4. Perform changes. The Change Report remains in the change reporter’s pending list. This process step is not supported by the CR document (PCMS*CTS functionality), but in terms of two Unix scripts.

(a) Extract DOCs. Each editor must extract the DOCs which he is responsible for. This is done by executing the Extract <CR-number> <editor> program.

(b) Edit DOCs. The documents are now in the workspaces of the different editors. They start text editors on the different DOCs and make changes according to the CR.

(c) Return DOCs. When finished, each editor must return all his DOCs to PCMS, and furthermore, action all returned DOCs from UNDER-WORK to DRAFT. This is done by executing the Return <CR-number> <editor> program.

5. Finish change report. When every affected document has been actioned to DRAFT the change responsible is allowed to action the CR to its next lifecycle state. The change responsible must:

(a) Select and open the CR from his pending list.

(b) Action CR from WORK (WORK phase) to DRAFT (FROZEN phase).

(c) Close the CR.

6. Approve documents. The CR enters the change manager’s pending list. The change manager must:

(a) Approve DOCs. This is done by executing the Approve <CR-number> program.

(b) Select and open the CR from his pending list.

(c) Action CR from DRAFT (FROZEN phase) to APPROVED (CLOSED phase).

(d) Close the CR.

6.6.6 Process Weaver – Change management

The Change management process, automated by means of Process Weaver works as follows:

1. Create change report. The change reporter starts the cooperative procedure called change.cp. The procedure prompts the change reporter for three input values, namely the PCMS database-id, the PCMS product-id, and the Workspace. The Workspace must be a directory owned by the weaver user, files extracted from PCMS are stored in this directory while they are updated. The change reporter then receives a work-context on his agenda. The change reporter must:

(a) Open the WC.

(b) Select an agent as change manager in a PW “selection list”.

174
(c) Select an agent as change responsible in a PW “selection list” (optional).

(d) Update the attributes revealing activity and estimated effort, implemented as PW “editable text area”.

(e) Write the problem description in a PW “editable text area”.

(f) Write affected DOCs in a PW “edit list” (initially empty). A complete list of all APPROVED documents is included in a PW “read-only document”. This list has been automatically generated by executing a PCMS query from the cooperative procedure CoShell code. Text from the PW “read-only document” can be copied into the PW “edit-list” (cut-and-paste).

(g) Press the WC “done” button.

2. Appoint change responsible. The change manager receives a work-context in his agenda. The change manager must:

(a) Open the WC.

(b) Read the CR, implemented as a PW “read-only document”.

(c) Select an agent to the change responsible role (a PW “selection list”). If an agent is already allocated he might approve the allocation, or change the allocation.

(d) Press the WC “done” button.

3. Find solution. The change responsible receives a work-context in his agenda.

(a) Approve solution. The change responsible must:

i. Open the WC.

ii. Update the proposed solution.

iii. Update the list of affected documents.

iv. Suspend the change management process if necessary. In this case he specifies a date when the process is to be resumed. A work-context is sent to the change manager with a message about the scheduled restart. The change manager may restart the process at any time by pressing the provided “continue” button. The process restarts on the next process step.

v. Press the WC “done” button.

(b) Distribute documents. The change responsible receives a work-context in his agenda. The change responsible must:

i. Open the WC.

ii. Select exactly one editor for each of the documents in the list of affected documents. This is achieved by cut-and-paste from a list of possible agents, into the list of affected documents.

iii. Press the WC “done” button.

4. Perform changes. Each editor receives one work-context for each of the documents he is responsible for. For each DOC, the editor must:

(a) Open the WC.

(b) Read the CR, implemented as a PW “read-only document”.

175
(c) Update the DOC, implemented as a PW “read-write document”. The DOC has automatically been extracted from PCMS.

(d) Press the WC “done” button. The DOC is automatically returned to PCMS and actioned to DRAFT.

5. Finish change report. When every affected document has been updated by the editors, a work-context arrives in the change responsible’s agenda. Every affected document has now automatically been returned to PCMS and actioned to DRAFT. The change responsible must:

(a) Open the WC.
(b) Read the CR, implemented as a PW “read-only document”.
(c) Press the WC “done” button. The CR is automatically actioned to DRAFT.

6. Approve documents. This process step is implemented as two work-contexts sent in sequence:

(a) Approve DOC. The change manager receives a work-context in his agenda. The change manager must:
   i. Open the WC.
   ii. Read the CR if necessary.
   iii. Press the WC “done” button. All DOCs are automatically actioned to AP-PROVED.

(b) Approve CR. The change manager receives a work-context in his agenda. The change manager must:
   i. Open the WC.
   ii. Read the CR if necessary.
   iii. Press the WC “done” button. The CR is automatically actioned to AP-PROVED.

6.7 Summary

This chapter has described the selection and implementation of two of the processes identified in the studied case, namely the Change management and Document review processes. Furthermore, it has described the functionality of the selected implementation environments, namely PCMS*CTS and Process Weaver. The process implementation has been described in terms of the three execution environments applied in the enactment experiment. The two processes and the three execution environments result in six possible combinations. Each of the six combinations has been described in detail.

The next Chapter will describe the experiment in terms of hypotheses, experimental design, results, and hypothesis evaluation. Chapter 8 will describe enactment experiences in terms of experiences with the applied tools, implemented processes, and repository integration.
Chapter 7

Experiment

Process automation is often advocated as a feasible way to improve the software development process. However, research in this area is hampered by the lack of experiments and empirical evidence. The main contribution in this Chapter to provide experimental results.

This Chapter describes a controlled experiment based on two processes identified in the case study part of the thesis. The involved processes and the three execution alternatives are described in the previous Chapter. The chapter is organized as follows: Section 7.1 describes hypotheses and experimental design. Section 7.2 describes results and evaluates the hypotheses. Section 7.3 discusses the generalization into project savings and yearly savings.

7.1 Hypotheses and experimental design

This section starts by identifying the factors that influence the experiment, both those that we deliberately vary and measure, and those that may disturb our results. On the background of these factors we state our hypotheses. Next, we describe the constraints that we had to take into account when we were designing the experiment. We then describe the chosen experimental design, and furthermore, how this design tries to eliminate the threats to internal validity. Finally, we discuss threats to the external validity of the experiment.

7.1.1 Factors in the design

We adapt the terminology used by Kish [82] where explanatory variables are divided into predictors and predictands. Other, extraneous sources of variation are called extraneous variables, and are divided into controlled variables, disturbing variables, and randomized variables.

The explanatory variables in our experiment were process, tool, and time. The two predictors were process and tool, and the predictand was time. The two predictors and their respective values are depicted in Table 7.1. The motivation for our explanatory variables is described in section 6.1 and section 6.2.

1The experimental design and analysis is performed together with Tor Stålhane and Svein Hallsteinsen, both SINTEF Informatics. I therefore use “we” and not “I” in this chapter.
<table>
<thead>
<tr>
<th></th>
<th>MANUAL PCMS</th>
<th>PCMS*CTS</th>
<th>Process Weaver</th>
</tr>
</thead>
<tbody>
<tr>
<td>Review</td>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>Change</td>
<td>D</td>
<td>E</td>
<td>F</td>
</tr>
</tbody>
</table>

Table 7.1: Six types of executions

The extraneous variables are threatening the validity of the experiment. According to Kish [82], one should always treat the extraneous variables as disturbing or randomized variables, and then try to move them into the set of controlled variables by appropriate actions.

The threat to validity can be divided into threat to internal validity and threat to external validity. Threats to internal validity are issues that may cause us to doubt the validity of our observations in the experiment itself. We identified the following threats:

- **Individual differences**: Individuals exhibit varying efficiency in carrying out trivial work and in using a keyboard, mouse and display interface. Also, the students carrying out the experiment had varying experience with the tools involved.

- **Learning**: As the experiment proceeds, the subjects become more and more familiar with the work and thus perform better.

We designed the experiment so that these effects were minimized, our design is described in section 7.1.4.

Threats to external validity are issues that cause problems when generalizing the observations into the real environment. We identified the following threats:

- **Representativeness of the environment**: The experiment was performed at the university and not at the Company site or by Company personnel. Furthermore, we only repeated the administrative work of previously executed processes, not the complete processes.

- **Representativeness of the selected process instances**: The experiment only involved a subset of the process instances, and furthermore, they were not randomly selected from the population.

These threats are discussed in section 7.1.5.

### 7.1.2 Hypotheses

This section describe our five hypotheses and their interpretation with respect to Table 7.1. The hypotheses are formalized and evaluated in section 7.2.

**PCMS*CTS support**

This hypothesis is about the relationship between the rectangles named (A,D) and those named (B,E) in Table 7.1.
\( \text{H}_0 \): There is no difference in the administrative overhead when comparing manually performed processes with ones that are supported by PCMS*CTS.

\textbf{Process Weaver support}

This hypothesis is about the relationship between the rectangles named (A,D) and those named (C,F) in Table 7.1.

\( \text{H}_0 \): There is no difference in the administrative overhead when comparing manually performed processes with ones that are supported by Process Weaver.

\textbf{Process independence of PCMS*CTS}

The term “degree of support” used below is defined as a percentage of their manually performed counterparts. E.g. a 50% reduction in the overhead when applying PCMS*CTS to both change and review processes will cause the degree of support to be equal.

This hypothesis is about the relationship between the rectangles named (A,B) and those named (D,E) in Table 7.1.

\( \text{H}_0 \): There is a difference in the degree of PCMS*CTS support when comparing the change and review processes.

\textbf{Process independence of Process Weaver}

This hypothesis is about the relationship between the rectangles named (A,C) and those named (D,F) in Table 7.1.

\( \text{H}_0 \): There is a difference in the degree of Process Weaver support when comparing the change and review processes.

\textbf{Tool comparison}

This last hypothesis came up during the implementation of the processes. It can be interpreted as tool flexibility, i.e. we believe a-priori that Process Weaver is more flexible in how the processes are implemented, or that Process Weaver realizations can more easily be made to reflect the original manual processes than the PCMS*CTS ones.

The hypothesis is about the relationship between the rectangles named (B,E) and those named (C,F) in Table 7.1.

\( \text{H}_0 \): There is no difference in the variation of overhead in process steps between PCMS*CTS and Process Weaver compared to their manual counterparts.
7.1.3 Constraints and limitations

The most important constraint to the experimental design was that the experiment had to take place at the University with students as subjects. This constraint was due to the Company who would not let the experiment take time from their development activities.

Furthermore, only a subset of the product database was released by the Company for use in the experiment. The implication was that we could not use processes referring to products outside this subset. This caused some problems with respect to the representativeness of the selected process instances.

There were six students in the group that implemented the two processes in Process Weaver and PCMS*CTS. The same group were used as subjects in the experiment. The experiment itself was not part of their project, so we had to negotiate the time to be used on the experiment. The group agreed to use one day for the experiment, and this limited the number of processes that could be executed.

The product database constraint excluded more than 90% of the original processes and the second one forced us to skip a few more, leaving us with 6 review processes and 7 change processes.

7.1.4 Experimental design

In order to control the effect of learning, all six experiments were carried out in parallel. In this way, although learning would still happen, it is reasonable to assume that all six experiments are subject to the same amount of learning.

In order to randomize the effects of individual differences, the students alternated between the tool and process combinations. Six workstations were used, one for each tool and process combination. To avoid access conflicts, each workstation had its own copy of the product database.

Initially, there was a pile of process descriptions next to each workstation, describing the processes to be carried out in the experiment. Each process description contained a detailed description of the procedure to be followed and a copy of the original change or review report. As the experiments proceeded, completed process steps were ticked off on the procedure descriptions.

Each student worked at one workstation, and changed workstation (and experiment) between each process step. Initially we planned for simple rotation, but deviated slightly from this scheme to avoid a lengthy process step in one of the experiments to block the progress of other experiments.

Prior to carrying out the experiment, the students spent one day of training to become familiar with the procedures and the tools and rules of the experiment.

Time was measured using a time measurement program on the workstations. This program allows recording time spent on a number of predefined tasks with an accuracy of one minute.
7.1.5 Threats to validity

By the experimental design described above, disturbing variables are reasonably well neutralized. However, we still have the problem of reality, that is; the degree to which the values measured are related to what we want to investigate.

Isolating administrative overhead

In real change and review processes at the Company the administrative tasks that we imitate in the experiments are interleaved with more creative work. Normally, change of both mindset and workstation context is necessary in both ends of a new task. In the experiments, the workstation always was in the right context and with the necessary tools up and running.

Since Process Weaver attempts to assist such shifts, the effect is probably that our experiment disfavors Process Weaver compared to a scenario with full fledged use of this tool as a workflow support tool.

Working skills

The developers at the Company perform these processes as part of their daily work and are familiar with them, while the students performing the process imitations for the experiments, learned this work only for the experiment. However, this is simple work, and after the one day training session, the students appeared to perform quite fluently. This effect can therefore be neglected.

Stability and response time of the tool environment

Neither stability nor response time was conceived by developers at the Company as hampering their work in any way. The same goes for the experiment, with one exception: Due to an installation problem, PCMS went down now and then. Fortunately there was no loss of data, so work could continue as soon as PCMS had been restarted. We do not think that this had any unfortunate effect on the realism of the experiment. On the contrary, it introduced arbitrary interrupts, which are quite common in real work situations.

Representativity of the selected processes

We cannot claim representativity on the basis of this selection process. To outweigh this weakness, we have tried to assess the representativity of the selection a posteriori in two ways:

- A subjective assessment by developers at the Company
- A comparison of characteristic parameters between the total population and the selection.
The assessment by Company developers concludes that the selection is representative of their projects. Corresponding characteristic parameters of the total population of processes and the selection is shown in Table 7.2.

<table>
<thead>
<tr>
<th></th>
<th>Review</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Project</td>
<td>Experiment</td>
</tr>
<tr>
<td>Authors</td>
<td>1.2</td>
<td>1.8</td>
</tr>
<tr>
<td>Documents</td>
<td>8.0</td>
<td>14.2</td>
</tr>
<tr>
<td>Effort</td>
<td>9.0</td>
<td>5.5</td>
</tr>
</tbody>
</table>

Table 7.2: Representativeness – process parameter values

Table 7.2 shows rather large differences between the total population and the subset used in the experiment. Thus, we cannot claim representativity of the selected subset. Furthermore, there are not enough data available to perform a proper impact analysis on the identified differences. However, we will discuss the impact on our results in an informal way:

- **Authors:** The number of authors will not have any impact because the administrative overhead in the two process implementations is the same for one and two authors.

- **Documents:** We expect to save relatively more administrative overhead as the number of documents increases, by using Process Weaver. We see from the table that the processes used in the experiment contain more documents than was generally true for the project processes. Thus, if we assume that Process Weaver performs relatively better as the number of documents increases, the experimental results will be too much in favor of Process Weaver. However, note that such conclusions cannot be drawn from the experimental results because of the low number of process executions.

- **Effort:** The effort used on each process instance does not influence the experimental results, but it influences the generalization into project savings (the relative importance of the administrative overhead). However, we solve this problem in section 7.3.1 by using the effort for the total population in our calculation of project savings.

To summarize, of the identified parameters the only parameter that we expect to have some effect is the number of involved documents. The expected effect is that Process Weaver will perform better on our subset than on an average review or change process. This will be further discussed in Section 7.2.5.

### 7.2 Results

In this section we will first present the measured values in terms of tables and bar-charts, and afterwards we will test our five hypotheses. Before testing the hypotheses we discuss methodological problems with respect to the statistical analysis. Furthermore we introduce some notation to make the statistical analysis more succinct and clear. At the end of the section we will discuss the problem of representativity on the background of the measured values.
7.2.1 Measured values

The data obtained from the experiment are shown in Table 7.4 and Table 7.5, page 191 and page 192.

<table>
<thead>
<tr>
<th></th>
<th>MANUAL PCMS</th>
<th>PCMS*CTS</th>
<th>Process Weaver</th>
</tr>
</thead>
<tbody>
<tr>
<td>Review</td>
<td>229</td>
<td>240</td>
<td>87</td>
</tr>
<tr>
<td>Change</td>
<td>291</td>
<td>232</td>
<td>107</td>
</tr>
<tr>
<td>Sum</td>
<td>520</td>
<td>472</td>
<td>194</td>
</tr>
</tbody>
</table>

Table 7.3: Aggregated results (minutes)

Table 7.3 shows the aggregated number of minutes used on each process and tool combination. E.g., the six review processes used a total of 229 minutes when they were performed manually (MANUAL PCMS). From this table we get an idea of the overall result: PCMS*CTS used a total of 472 minutes compared to 520 minutes of the manual process, 48 minutes less than the manual process (9% better). Process Weaver used a total of 194 minutes compared to 520 minutes of the manual process, 326 minutes less than the manual process (63% better).

![Figure 7.1: Document review - aggregated results](image)

Figure 7.1 and Figure 7.2 show a graphical representation of the experimental results. The process support results can be seen by comparing the three bars in each of the two figures. The process independence results can be seen by comparing the two figures. The tool comparison result can be seen in the bar-charts by comparing the shaded areas, one pattern for each process step. The remaining sections will present the statistical analysis and test the hypotheses.

7.2.2 Methodological problems

Before testing the hypotheses set forth in section 7.2.4, we need to decide which assumptions - if any - we can build our analysis on. The most important of these assumptions is the
possible assumption of normality. We will argue as follows\(^2\):

The total time for each job in each work process is a sum of several durations, presumably decided by the work step at hand more than by this particular review or change process. Under rather general assumptions, a sum of independent variables will be normally distributed as long as none of the terms in the sum has a much larger variance than the others. This result follows from Levy-Lindbergh's central limit theorem.

Based on this, it is reasonable to assume that the total time for each job is normally distributed. In order to check this assumption, we ran the Shapiro-Wilk's test on all data sets. In no case could we reject the hypothesis of normal distribution.

It may seem unreasonable to use the normal distribution for a variable that can not take on negative values. It is possible to deal with this objection by using a truncated normal distribution. The correction needed, however, turns out to be negligible and we have thus used the normal distribution as a reasonable approximation to the observed data. However, in order not to depend entirely on this assumption, we have in addition — whenever possible — used non-parametric methods to check on the results.

We have thus used the T-statistics for hypotheses concerning equality of job-durations and checked the results by means of the Friedman statistics.

For the hypotheses concerning ratios, it has not been possible to find a non-parametric test and these results are thus depending on the normal approximation alone.

In addition, the following complication needs to be considered: Many of the hypotheses are concerned with relations between the jobs performed with for instance the Process Weaver and the same jobs done manually. Our data, however, are concerned with two different types

\(^2\)The statistical argumentation and analysis is performed by Tor Stålhane, SINTEF Informatics, Trondheim, Norway.
of jobs – reviews and change management respectively. Thus, each hypothesis must be split into two hypotheses and the confidence level of the combined hypothesis is the product of the two confidence levels. If we for instance accept one of the two sub-hypotheses with a 96% confidence and the other with a 98% confidence then the combined hypothesis will have a confidence of 94% only.

7.2.3 Some notations

We will use the following notational conventions:

- $\mu(t,p)$ is the expected value for the overhead for process $p$ when using the tool $t$. For the sake of simplicity, performing the process manually will be denoted by the "tool" $M$.
- $\sigma(t,p)$ is the standard deviation for the overhead for process $p$ when using the tool $t$.
- "*: means "any process" or "any tool".
- $T(t,p)$ is the overhead for process $p$ when using the tool $t$. When it is necessary, we will use a subscript to indicate that the duration time is related to a special instance of the process $p$. This instance is denoted $T_i(t,p)$.
- $T(t,p,k)$ is the overhead for process $p$, step number $k$ when using the tool $t$. Each process step has a distinct identifier. However, since they do not matter here, we have chosen to use integers from 1 to 6 in all cases.
- $X_i(t,p)$ is the relative saving – time reduction – between tool $t$ and the process performed manually for process instance $i$.

Based on the already performed Shapiro-Wilk’s tests we can furthermore write: $T(t,p) \sim \text{Normal}(\mu(t,p), \sigma^2(t,p))$

7.2.4 Testing the hypotheses

Before concluding on each hypothesis, we will reformulate it, using standard mathematical notations and the conventions described in the previous section. This is done in order to be clear about what really is the issue in each case.

PCMS*CTS Support

$H_0 : \mu(M,.) = \mu(\text{PCMS}_i,.)$

The hypothesis of no difference between a manual process and a process supported by the PCMS*CTS is accepted at the 95% level. This result is confirmed by the non-parametric Friedman test.
Process Weaver support

\[ H_0 : \mu(M, \cdot) = \mu(PW, \cdot) \]

The hypothesis of no difference can be rejected with 94% confidence – 99.9% for the change management and 94.1% for the review process.

If we use the Friedman statistics we find that we can reject the hypothesis with a 99.5% confidence. Thus, even if the result from the T-statistics is a little on the bad side – 94.1% instead of 95% – we will reject the hypothesis of no difference.

The major reason for Process Weaver being “better” is that the process programmer has fewer restrictions compared to those in the PCMS*CTS environment. Information was entered once and then kept while it was needed, while it had to be reentered in the PCMS*CTS case. Check-in and check-out was performed when needed and lifecycle status set automatically.

With PCMS*CTS one had to go through complicated menu hierarchies to access the simplest attribute. In the Process Weaver implementation attribute values could be keyed in directly, bypassing such menu hierarchies.

Process independence of PCMS*CTS

Let \( X_i(t, p) = T_i(t, p)/T_i(M, p) \) for each individual process instance \( i \) and denote the average value of \( X \) as \( \Xi \). We can then re-state our hypothesis as:

\[ H_0 : \Xi(PCMS, DR) > \Xi(PCMS, CM) \]

In order to check this hypothesis we computed the ratio between the total durations for each job done manually and by using PCMS*CTS for support. The resulting two data sets pass the Shapiro-Wilk’s test for normality. Based on the T-statistics, and with a significance level of 95%, we will accept the hypothesis of difference in degree of PCMS*CTS support for the two processes. PCMS*CTS supports the Change management process better than it supports the Document review process.

Our explanation is that the PCMS*CTS tool is built for change processes, i.e., processes in which it is mandatory to create new revisions for all documents such as the Change management process. On the other hand, the document review process had an identified set of documents with an unknown faith. After the review meeting each document receives a review result that decides the next step of that document, the consequence might be different numbers of revisions for different documents (none, one, or many).

Process independence of Process Weaver

\[ H_0 : \Xi(PW, DR) > \Xi(PW, CM) \]

Also here, we computed also here the relevant ratios and used the T-statistics. Based on the T-statistics, we reject the hypothesis of difference in the degree of savings relative to the manually performed process between reviews supported by Process Weaver and change management supported by Process Weaver.
Thus, we can conclude that Process Weaver provides more general process support than PCMS*CTS. Process Weaver is less sensitive to specific process characteristics than PCMS*CTS, this because the process programmer is offered more freedom in the Process Weaver case.

**Tool comparison**

We have decided to reformulate the hypothesis in question as follows:

\( H_{01} \): The savings when moving from a manual process to a process supported by PCMS is caused by savings in a few areas while there are no savings for most of the other \( k \) steps

\[ T(M, DR, k) = T(PCMS, DR, k) \text{ and } T(M, CM, k) = T(PCMS, CM, k) \]

\( H_{02} \): The savings when moving from a manual process to a process supported by Process Weaver is caused by savings in a few areas while there are no savings for most of the other \( k \) steps

\[ T(M, DR, k) = T(PW, DR, k) \text{ and } T(M, CM, k) = T(PW, CM, k) \]

We will test both these hypothesis by using the Friedman non-parametric test for difference between two series of values. This gives us the following results:

\( H_{01} \): For DR – review – and for CM – change management – we accept the hypothesis for PCMS.

The PCMS*CTS implementation redistributes effort on process steps compared to the manually performed process. We interpret this as lack of flexibility, that the tool strongly influences on how the process can be performed. As an example we may look at the shaded areas in Figure 7.1. We see that “Update review report” takes more time with PCMS*CTS support than in the manual case, furthermore, that the opposite is true for “Correct and control documents”.

\( H_{02} \): For DR – review – we reject the hypothesis for PW. For CM – change management – we accept the hypothesis for PW.

The Process Weaver implementation of the review process saves time in most process steps. This is interpreted as being due to a high level of flexibility, because it saves effort in different types of work. For the change management this effect cannot be identified.

### 7.2.5 Comments on representativeness

In Section 7.1.5 we stated that we expected Process Weaver to perform better on our subset than on an average review or change process. However, augmented by the experimental results, we expect this effect to be small. This because of the following argument:

- The worst case review process is DR25 with one document. The savings in this process can be calculated to 61.1% from the numbers found in Table 7.4. The total savings of all six review processes are 62%.
• The worst case change process is CM34 with two documents. From Table 7.5 we find the savings in CM34 to be 65.4% and total savings of all the seven change processes to be 63.2%.

We see that for the review process, Process Weaver performs slightly better in average than in the worst case. For the change process the opposite is true, Process Weaver performs better in the worst case than in the average case. In both cases the savings are about 60%, and we can conclude that the savings are not very sensitive to the number of documents. Thus, the differences between the selected processes and the total population are not important.

7.3 Discussion

In this section we will generalize the results into savings in this particular Company project. We will also give some comments on generalization into Company projects and to projects in general. Furthermore, we will outline the lessons learned on PCMS*CTS and Process Weaver. Finally, we will describe factors that are not considered in our experiment.

7.3.1 Project savings

The statistical analysis in section 7.2 shows that the savings when applying Process Weaver are not likely to be accidental. Furthermore, that the hypothesis that PCMS*CTS will save work can be rejected with 95% confidence. Thus, we believe that we can save administrative effort by using Process Weaver. What is more interesting, however, is the actual effort saved in the project.

We use the arithmetical average of execution times in the change and review processes, respectively. This results in a saving of 23.7 minutes in each review process and 33.2 minutes in each change process. We multiply this with the number of processes which results in 1682.7 minutes for review and 3386.4 minutes for review. The total savings in these two processes then becomes 5069.1 minutes or 84.5 hours. The total effort in the project was 7055 person-hours, so the Process Weaver application reduces the project effort by 1.2%.

7.3.2 Generalizing into projects

The selected processes are not unique to this project, they have been performed for the last couple of years and are likely to be used in that way for the foreseeable future. Thus, we expect the savings to be approximately 30 minutes for each process execution in future projects. However, the execution frequency of the two processes may vary from one project to another, and thus the total savings will vary. E.g. the change management process depends, among other things, on the requirements stability.

The numbers from our experiments cannot be generalized to other companies and tool environments. However, we feel confident that processes with an administrative overhead can be automated such that the overhead is significantly reduced. Process automation should be considered for processes that are executed often, or have large administrative overheads. As
a basis of comparison, the reader can use the two processes that are described in this chapter. The change management process was executed 102 times with an administrative overhead of 5.0%. The document review process was executed 71 times with an administrative overhead of 7.1%. By using Process Weaver we reduced the administrative overhead with 63% in the change process and 62% in the review process.

7.3.3 Return of investments

We do not have sufficient data from other projects to assess the representativity of the number of change and review processes. However, the subjective opinion of Company personnel is that the studied project is representative. On the average there are three such projects in the Company each year. Thus, the yearly savings of using Process Weaver is approximately 240 person-hours. With a price of $70 for each hour of work we have a yearly cost reduction of approximately $17,000.

We cannot expect the foreseeable future to be more than three years, thus the return of investments should be manifested in this time-frame. The cost of the initial implementation can be divided into the cost of the Process Weaver environment, installation, training, and process implementation. The cost of Process Weaver with 15 concurrent users is approximately $20,000. Our experience from the student project is that installation and training costs are about 200 person hours, corresponding to $11,000 of Company costs. The process implementation, including the PCMS interface, took another 200 hours. In addition to these fixed costs ($41,000) there is a cost each year for Process Weaver administration as well as small fixes and changes to process implementations. We expect a need for a part-time Process Weaver administrator using at least 100 hours each year on this type of work ($7,000). The expenses in three years become $55,000, $41,000 in the initial year, and $14,000 in the following two years. The expected savings are $51,000, $17,000 each year. Thus, we can expect a loss of $3000.

This cost/benefit analysis shows that even if Process Weaver significantly reduces the administrative overhead in the two processes, it is not likely that the benefits outweigh the costs.

If we ignore the costs associated to purchase, installation, training, and maintenance of the tools – and only look at process programming costs – we get the following picture: While the cost of implementing the processes in PCMS*CTS is almost insignificant (10 person-hours), the cost of implementing the processes in Process Weaver is rather high (200 person-hours). The experiment shows savings of approximately 0.5 hour on each process instance, thus about 400 instances need to be executed before we reach break even.

7.3.4 Potential benefits not considered in the experiment

Beside reduced administrative overhead, process technology might provide benefits such as reduced number of process failures, increased process conformance, and increased quality and availability of process data. On the other hand, process technology might provide drawbacks such as low acceptance among process participants, inflexibility, and not being compatible with other technologies. It might also have missing or erroneous functionality and thus
represent a risk to the organization. These issues will be discussed in more details in the next chapter.

7.4 Summary

In this chapter I have reported experimental results regarding process automation. In the experiment we performed two processes in three different ways while measuring the time used on administrative overhead. Our hypotheses are tested in section 7.2.4. The main results are:

- Process Weaver support will reduce the administrative overhead in the two processes by 60%. PCMS*CTS support will have no effect.
- Process Weaver support will be equally good in both processes. PCMS*CTS support will be different for the two processes.
- Process Weaver support will be more adaptable to organizational procedures than is the case for PCMS*CTS.

Furthermore, I generalized the results into savings in the project in which the processes had been executed. The savings in this project would have been 84.5 person hours, or 1.2% of overall project effort if the project had used the Process Weaver implementation.

Finally, I discussed return of investments, and concluded that it is probably not cost effective to introduce Process Weaver into the studied organizational environment for the purpose of reducing project effort. If one only considers the process programming costs, I expect that a cost break even will be reached after executing 400 process instances.
<table>
<thead>
<tr>
<th>Sub-Task / Task</th>
<th>DR62</th>
<th>DR30</th>
<th>DR15</th>
<th>DR25</th>
<th>DR23</th>
<th>DR19</th>
<th>Sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manual</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>00:22</td>
<td>00:20</td>
<td>00:25</td>
<td>00:09</td>
<td>00:16</td>
<td>00:08</td>
<td>01:40</td>
</tr>
<tr>
<td>2</td>
<td>00:06</td>
<td>00:07</td>
<td>00:07</td>
<td>00:04</td>
<td>00:06</td>
<td>00:03</td>
<td>00:33</td>
</tr>
<tr>
<td>3</td>
<td>00:01</td>
<td>00:03</td>
<td>00:09</td>
<td>00:01</td>
<td>00:04</td>
<td>00:02</td>
<td>00:20</td>
</tr>
<tr>
<td>4</td>
<td>00:01</td>
<td>00:15</td>
<td>00:34</td>
<td>00:02</td>
<td>00:02</td>
<td>00:04</td>
<td>00:58</td>
</tr>
<tr>
<td>5</td>
<td>00:01</td>
<td>00:03</td>
<td>00:03</td>
<td>00:01</td>
<td>00:02</td>
<td>00:01</td>
<td>00:11</td>
</tr>
<tr>
<td>6</td>
<td>00:02</td>
<td>00:01</td>
<td>00:01</td>
<td>00:01</td>
<td>00:01</td>
<td>00:01</td>
<td>00:07</td>
</tr>
<tr>
<td>Sum</td>
<td>00:33</td>
<td>00:49</td>
<td>01:19</td>
<td>00:18</td>
<td>00:31</td>
<td>00:19</td>
<td>03:49</td>
</tr>
<tr>
<td>PCMS*CTS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>00:14</td>
<td>00:26</td>
<td>00:31</td>
<td>00:08</td>
<td>00:13</td>
<td>00:13</td>
<td>01:45</td>
</tr>
<tr>
<td>2</td>
<td>00:17</td>
<td>00:14</td>
<td>00:23</td>
<td>00:05</td>
<td>00:12</td>
<td>00:05</td>
<td>01:16</td>
</tr>
<tr>
<td>3</td>
<td>00:01</td>
<td>00:00</td>
<td>00:01</td>
<td>00:00</td>
<td>00:02</td>
<td>00:00</td>
<td>00:04</td>
</tr>
<tr>
<td>4</td>
<td>00:01</td>
<td>00:06</td>
<td>00:09</td>
<td>00:01</td>
<td>00:00</td>
<td>00:02</td>
<td>00:19</td>
</tr>
<tr>
<td>5</td>
<td>00:02</td>
<td>00:06</td>
<td>00:14</td>
<td>00:02</td>
<td>00:02</td>
<td>00:03</td>
<td>00:29</td>
</tr>
<tr>
<td>6</td>
<td>00:01</td>
<td>00:01</td>
<td>00:02</td>
<td>00:01</td>
<td>00:01</td>
<td>00:01</td>
<td>00:07</td>
</tr>
<tr>
<td>Sum</td>
<td>00:36</td>
<td>00:53</td>
<td>01:20</td>
<td>00:17</td>
<td>00:30</td>
<td>00:24</td>
<td>04:00</td>
</tr>
<tr>
<td>Process Weaver</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>00:07</td>
<td>00:07</td>
<td>00:09</td>
<td>00:03</td>
<td>00:07</td>
<td>00:03</td>
<td>00:36</td>
</tr>
<tr>
<td>2</td>
<td>00:06</td>
<td>00:06</td>
<td>00:13</td>
<td>00:02</td>
<td>00:01</td>
<td>00:02</td>
<td>00:30</td>
</tr>
<tr>
<td>3</td>
<td>00:00</td>
<td>00:00</td>
<td>00:00</td>
<td>00:00</td>
<td>00:00</td>
<td>00:00</td>
<td>00:00</td>
</tr>
<tr>
<td>4</td>
<td>00:00</td>
<td>00:05</td>
<td>00:08</td>
<td>00:01</td>
<td>00:00</td>
<td>00:01</td>
<td>00:15</td>
</tr>
<tr>
<td>5</td>
<td>00:00</td>
<td>00:00</td>
<td>00:00</td>
<td>00:00</td>
<td>00:00</td>
<td>00:00</td>
<td>00:00</td>
</tr>
<tr>
<td>6</td>
<td>00:01</td>
<td>00:01</td>
<td>00:02</td>
<td>00:01</td>
<td>00:00</td>
<td>00:01</td>
<td>00:06</td>
</tr>
<tr>
<td>Sum</td>
<td>00:14</td>
<td>00:19</td>
<td>00:32</td>
<td>00:07</td>
<td>00:08</td>
<td>00:07</td>
<td>01:27</td>
</tr>
</tbody>
</table>

Table 7.4: Experimental results from the Review process
<table>
<thead>
<tr>
<th>SUB-TASK/TASK</th>
<th>CM31</th>
<th>CM52</th>
<th>CM32</th>
<th>CM44</th>
<th>CM30</th>
<th>CM34</th>
<th>CM23</th>
<th>Sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>MANUAL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>00:08</td>
<td>00:06</td>
<td>00:06</td>
<td>00:06</td>
<td>00:04</td>
<td>00:08</td>
<td>00:01</td>
<td>00:39</td>
</tr>
<tr>
<td>2</td>
<td>00:01</td>
<td>00:00</td>
<td>00:01</td>
<td>00:00</td>
<td>00:01</td>
<td>00:01</td>
<td>00:00</td>
<td>00:04</td>
</tr>
<tr>
<td>3</td>
<td>00:14</td>
<td>00:11</td>
<td>00:17</td>
<td>00:09</td>
<td>00:15</td>
<td>00:06</td>
<td>00:18</td>
<td>01:30</td>
</tr>
<tr>
<td>4</td>
<td>00:05</td>
<td>00:03</td>
<td>00:03</td>
<td>00:03</td>
<td>00:00</td>
<td>00:03</td>
<td>00:02</td>
<td>00:19</td>
</tr>
<tr>
<td>5</td>
<td>00:03</td>
<td>00:06</td>
<td>00:06</td>
<td>00:04</td>
<td>00:05</td>
<td>00:02</td>
<td>00:09</td>
<td>00:35</td>
</tr>
<tr>
<td>6</td>
<td>00:13</td>
<td>00:12</td>
<td>00:16</td>
<td>00:10</td>
<td>00:15</td>
<td>00:06</td>
<td>00:32</td>
<td>01:44</td>
</tr>
<tr>
<td>Sum</td>
<td>00:44</td>
<td>00:38</td>
<td>00:49</td>
<td>00:32</td>
<td>00:40</td>
<td>00:26</td>
<td>01:02</td>
<td>04:51</td>
</tr>
<tr>
<td>PCMS*CTS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>00:07</td>
<td>00:06</td>
<td>00:05</td>
<td>00:05</td>
<td>00:05</td>
<td>00:06</td>
<td>00:04</td>
<td>00:38</td>
</tr>
<tr>
<td>2</td>
<td>00:03</td>
<td>00:02</td>
<td>00:02</td>
<td>00:02</td>
<td>00:02</td>
<td>00:02</td>
<td>00:01</td>
<td>00:14</td>
</tr>
<tr>
<td>3</td>
<td>00:21</td>
<td>00:16</td>
<td>00:18</td>
<td>00:14</td>
<td>00:07</td>
<td>00:11</td>
<td>00:32</td>
<td>01:59</td>
</tr>
<tr>
<td>4</td>
<td>00:02</td>
<td>00:02</td>
<td>00:01</td>
<td>00:01</td>
<td>00:01</td>
<td>00:01</td>
<td>00:01</td>
<td>00:09</td>
</tr>
<tr>
<td>5</td>
<td>00:02</td>
<td>00:03</td>
<td>00:02</td>
<td>00:01</td>
<td>00:02</td>
<td>00:01</td>
<td>00:02</td>
<td>00:13</td>
</tr>
<tr>
<td>6</td>
<td>00:06</td>
<td>00:04</td>
<td>00:08</td>
<td>00:06</td>
<td>00:06</td>
<td>00:04</td>
<td>00:05</td>
<td>00:39</td>
</tr>
<tr>
<td>Sum</td>
<td>00:41</td>
<td>00:33</td>
<td>00:36</td>
<td>00:29</td>
<td>00:23</td>
<td>00:25</td>
<td>00:45</td>
<td>03:52</td>
</tr>
<tr>
<td>Process Weaver</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>00:02</td>
<td>00:03</td>
<td>00:03</td>
<td>00:04</td>
<td>00:04</td>
<td>00:03</td>
<td>00:04</td>
<td>00:25</td>
</tr>
<tr>
<td>2</td>
<td>00:01</td>
<td>00:00</td>
<td>00:01</td>
<td>00:01</td>
<td>00:02</td>
<td>00:00</td>
<td>00:00</td>
<td>00:05</td>
</tr>
<tr>
<td>3</td>
<td>00:09</td>
<td>00:08</td>
<td>00:07</td>
<td>00:05</td>
<td>00:06</td>
<td>00:04</td>
<td>00:04</td>
<td>00:43</td>
</tr>
<tr>
<td>4</td>
<td>00:01</td>
<td>00:01</td>
<td>00:01</td>
<td>00:00</td>
<td>00:00</td>
<td>00:00</td>
<td>00:00</td>
<td>00:03</td>
</tr>
<tr>
<td>5</td>
<td>00:01</td>
<td>00:01</td>
<td>00:01</td>
<td>00:00</td>
<td>00:01</td>
<td>00:01</td>
<td>00:01</td>
<td>00:06</td>
</tr>
<tr>
<td>6</td>
<td>00:05</td>
<td>00:03</td>
<td>00:03</td>
<td>00:02</td>
<td>00:07</td>
<td>00:01</td>
<td>00:04</td>
<td>00:25</td>
</tr>
<tr>
<td>Sum</td>
<td>00:19</td>
<td>00:18</td>
<td>00:16</td>
<td>00:12</td>
<td>00:20</td>
<td>00:09</td>
<td>00:13</td>
<td>01:47</td>
</tr>
</tbody>
</table>

Table 7.5: Experimental results from the Change management process
Chapter 8

Enactment experiences

The enactment experiment provided experimental results with respect to the applied tools and the implemented processes. Furthermore, the implementation process provided experiences with processes and tools which are not experimental results. This Chapter describes these experiences in terms of a comparison between Manual PCMS, PCMS*CTS and Process Weaver, and identified weaknesses in PCMS*CTS and Process Weaver.

This Chapter is organized as follows: Section 8.1 compares the three execution environments with respect to identified properties. Section 8.2 reports strong and weak features in PCMS*CTS. Section 8.3 reports strong and weak features in Process Weaver.

8.1 Comparing the execution environments

This section compares the three execution environments with respect to identified properties. An overview of the comparison is presented in Table 8.1, and each of the properties are discussed in detail in the following sections. The symbols used in Table 8.1 are (−), (•), (**), and (***), the first symbol means that no process support is identified, the other being a ranking of the three environments, (***) being the high value.

8.1.1 Chaining and notification

Chaining and notification: Notification; the extent to which process agents are made aware of process events. Chaining; the extent to which process steps and sequencing among process steps can be defined.

Manual PCMS offers chaining in the sense that agents associated to the next role in the lifecycle of an item revision receives an e-mail when the item is actioned. This functionality is associated to the product and not the process because PCMS has no knowledge about the process context. However, in cases where the process involves a single document, the process can be implemented as an item lifecycle, thus, some weak process support is in place.

PCMS*CTS offers chaining in the sense that agents associated to the next role in the lifecycle
of a change document receives an e-mail as well as the change document itself, when the change document is actioned. The change document enters the pending list of all “next” agents. This functionality is associated to the process because the change document represents the process. The chaining and notification is defined in terms of the change document’s lifecycle. The chaining is limited because concurrent threads are not supported.

Process Weaver offers general chaining in the sense that it can be fully and freely defined by the process programmer. The process chaining is defined in the cooperative procedure, sending work-contexts to process agents (notification), and waiting for answers from work-contexts previously sent. Any number of concurrent threads can be defined.

A related property is programming flexibility. Seen from the point of view of the process programmer, he can create a wider range of process programs with Process Weaver than with PCMS*CTS. As a consequence he can create programs that are closer to the existing manual process. However, this view must be balanced with a discussion of enforcement, flexibility seen from the user’s point of view (see below). Another related property is coordination support, in which notification supports event coordination, while chaining supports process coordination. Data coordination is supported by repository and workspace functionality.

### 8.1.2 Tailored work environments

Tailored work environments: Work environment; the process interface offered to process agents. Tailored work environment; the extent to which the work environment can be adapted to the process or individual process steps.

Manual PCMS offers the “product browser” to navigate in the repository, create and extract items, update attributes, etc. An extracted item is locked for other agents until it is returned. Thus, Manual PCMS offers “standard” repository functionality, and no defined process. The repository functionality can also be used in the PCMS*CTS and Process Weaver execution environments.

PCMS*CTS offers the change document as an interface to the process. The change document is tailored to a process type, by means of a change document type. The change document
type is defined by the process programmer in terms of content and layout. Although the change document can be tailored to specific lifecycle roles, its tailoring is less general than in Process Weaver.

Process Weaver offers work contexts as an interface to the process. A work-context is defined by the process programmer in terms of content and lay-out. The Process Weaver work-context is more tailorable than change documents because two work-contexts used within a cooperative procedure are independently defined, while there is exactly one change document in a PCMS*CTS process.

8.1.3 Workspace functionality

Workspace functionality: Workspace; the place were documents are stored while they are being edited. Workspace functionality; the extent to which the environment offers functionality to control the presence of – and access to – documents in the workspace.

The workspaces in all execution environments are Unix directories. In the cases of Manual PCMS and PCMS*CTS the process agents extract files from the repository by means of the PCMS “product browser”. The difference is that with PCMS*CTS the “process reports” are not extracted because they are implemented as change documents. This can be seen as PCMS*CTS offering a workspace for “process reports”, but not for “process documents”.

In the Process Weaver case both “process reports” and “process documents” are extracted similarly to manual PCMS. However, in the Process Weaver case, the repository extracts and returns are not seen by the process agents but it is embedded in the process program. The process agents only see a document loaded into an editor when they open a work context document symbol. Because items are extracted by the “weaver” Unix user, the files are owned by “weaver”. The process programmer have the possibility and responsibility to control when the items are in the workspace(s), and how they can be updated by process agents. The latter is not well supported by Unix because one cannot grant access to individual Unix users, furthermore, only the Unix root user (and not “weaver”) can change ownership of files and directories. To summarize, Process Weaver does not offer a workspace concept, but the process programmer may implement workspace functionality by means of the cooperative procedure and work contexts.

8.1.4 Process automation

Process automation: The extent to which process steps can be performed without involving the process agent.

Manual PCMS and PCMS*CTS do not offer process automation capabilities. Process Weaver offers automation since the process programmer may write general code in the cooperative procedure, this code can also access programs external to the cooperative procedure. In our case we automatically performed PCMS operations. The automation is restricted by the problem domain, not by the Process Weaver environment. The problem domain is what is needed or wanted by the process owner. E.g., we did not implement the document control because process owners decided it was better left unsupported. However, note that general
code cannot be associated to the work context (discussed in the Process Weaver evaluation).

8.1.5 Process enforcement

Process enforcement: *The extent to which the process program limits the process choices made by process agents.*

Manual PCMS only limits product choices, not process choices. The agent’s product choices are limited by the locking of extracted items, by the access rights associated to parts and roles, and by the enforced creation of new revisions when the item is not in its initial lifecycle state.

PCMS*CTS limits the process choices by controlling the access to the change document. This is controlled by the change document’s lifecycle (states and roles). Only process agents with the change document in their pending list can access the change document, this can be further restricted by defining a leader agent within a change document role. Furthermore, the change document limits the process choices by the phase concept, e.g., new items cannot be defined as affected when the change document is in its work phase. When used together with change documents, item operations can be limited by using flags on item lifecycle states. Finally, the content and interaction with the change document is limited by the defined change document type. However, a change document does not limit the access to affected documents, as any agent with the proper item lifecycle role may extract an affected document while the change document is in its work phase.

Process Weaver limits the process choices in terms of the implemented cooperative procedures and work-contexts. The process programmer defines the level of freedom in the implementation, but seen from the viewpoint of the involved agents, the programmed set of choices is enforced. In general, one can implement a higher degree of enforcement with Process Weaver than with PCMS*CTS. E.g., one can enforce the distribution of affected documents among process agents with Process Weaver, but not with PCMS*CTS. However, in some cases, it is impossible — or at least difficult — to enforce constraints among repository items by means of a cooperative procedure. E.g., PCMS*CTS enforces that an affected item cannot be extracted in the change document’s analysis phase, and that an item with a change document flag on a lifecycle state cannot be actioned past this state without being listed as affected in a change document.

8.1.6 Process history

Process history: *The extent to which the process program collects data about the process and makes them formatted and accessible by external programs.*

Manual PCMS supports data gathering by means of item revisions and the item history. Each time an item is actioned, PCMS stores transition, time, and agent information about the event. To the extent that the item lifecycle can be considered as a kind of weak process support, it must also be accepted as process history support. This information is accessible by PCMS reports or by external programs through the PCMS/API. This information will also be stored in the PCMS*CTS and Process Weaver implementations. However, in the Process
Weaver case this information is degraded because the “weaver” user will be registered as the agent in all cases, i.e. since “weaver” extracts, returns, and actions items.

PCMS*CTS supports data gathering through the change document in three ways: First, the change document references affected documents by means of repository relationships. These relationships can be queried and used for reporting and data analysis, by PCMS reports or by external programs. Second, an action history entry is created each time the change document is actioned. Furthermore, a change document can be regenerated with its original content at any historical action point. Third, an attribute history entry is created whenever the attribute value changes. Thus, the complete history of a change document is maintained by PCMS*CTS. Note that the change document history comes for free, there are no process implementation costs.

Process Weaver does not provide any automatic data gathering related to cooperative procedure and work contexts\(^1\). However, since general code can be written in cooperative procedures, the process programmer may implement process history, e.g., by logging events to a file. Note that this comes at a significant implementation cost since the log file must have a defined format, program functions to store events must be implemented, and function invocation must be correctly distributed in the cooperative procedure code.

Finally, some comments on the Review and Change reports as implemented in the experiment. One can view the created Review and Change reports (“process reports”) as process history, as opposed to the affected documents which are seen as product history. In the Manual PCMS and Process Weaver cases, when the Change management and Document review processes finish, the “process reports” contain information according to their last update. In the PCMS*CTS case they are change documents and will contain the complete process history.

8.1.7 Process queries and reports

Process queries and reports: *The extent to which the process agents can access information about current and historical process instances.*

Manual PCMS supports a wide range of *product* reports, including reports about part structuring, item revisions, and item history. These reports are also available in the PCMS*CTS and Process Weaver implementations. The product information is accessible through the PCMS/API. However, to the extent that the item lifecycle can be considered as a kind of weak process support, it must also be accepted as process report support.

PCMS*CTS offers a wide range of *process* reports, divided into pending and catalog reports. The pending reports cover change documents which are currently active in the system, and catalog reports cover all change documents in the system.

Process Weaver does not support process queries, since no process history is maintained by Process Weaver. If the process history were programmed into the cooperative procedure, the process programmer must still implement query or report functionality on the created log files.

\(^1\)The Process Weaver Activity Instance Manager (AIM) supports data gathering, but the AIM is not within the scope of the current discussion.
8.1.8 Process documents

Process documents: The extent to which the identification and registration of affected documents are supported.

Manual PCMS does not support the identification and registration of affected documents. With Manual PCMS the item identifier had to be manually keyed into the “process report” as a text string. If the item identifier is not known, the process agent must browse the product repository to look for alternatives.

PCMS*CTS offers support to identify and register the process documents. The responsible agent may select items to be related as “affected” directly from the change document. The agent can specify a filter on the PCMS repository and then the change document lists the possible alternatives. The agent simply selects the “affected items” from this list. He may also unrelate affected items in case of mistakes.

Process Weaver does not support the identification and registration of affected documents as such, but the identification can be supported by the process program by executing queries on repository information. The registration cannot be supported, essentially the item identifier has to be manually keyed into a PW “edit list” as a text string.

The PCMS*CTS approach takes less time and is less annoying to process agents. However, what is more important is that the creation and maintenance of explicit relationships causes the information to be accessible for reporting and analysis. E.g., these relationships were utilized in a set of Unix scripts which automatically extracted affected documents into the workspace of the responsible agent.

8.1.9 Role binding

Role binding: The extent to which the environment supports the allocation of agents to roles.

Manual PCMS does not support role binding. In this execution environment the agents were allocated to the role fields of the change and review reports.

PCMS*CTS automatically assigns agents to a change document’s static roles according to bindings in the product’s part hierarchy. The change document’s dynamic roles are allocated by process agents by applying the PCMS*CTS “delegate” dialogue. The process agent selects a role and then selects one of the agents in the specified “candidate list”. The “candidate list” is associated to the change document type.

Process Weaver’s cooperative procedures treat a role as a CoShell variable which may receive a value from a “selection list” or “edit text area”. The variable’s value is used when the cooperative procedure sends a work context to an agent. All other strategies must be developed by the process programmer, e.g., reading a file with role information, or query the PCMS repository.

8.1.10 Development costs

Development costs: A high value means that it is costly to implement the process support.
To the extent that Manual PCMS can be denoted as process support, it is free in the sense that items must have lifecycles. The cost of defining a lifecycle is considered insignificant.

PCMS*CTS provides a rather strict set of features, but benefits from low implementation costs. Process Weaver gives the process programmer a higher level of freedom, but at the price of high implementation costs. In preparing the experiment, the project group used about 200 hours on the Process Weaver implementations, compared to 20 hours on the PCMS*CTS implementations. Furthermore, PCMS*CTS comes with the PCMS system at no additional cost, and it requires little additional training with respect to what is needed to use PCMS in the first place.

### 8.2 PCMS*CTS experiences

All strong features of the PCMS*CTS environment can be related to the tight integration with the PCMS repository, e.g., its capability to: Relate affected documents, enforce lifecycle constraints, maintain the process history, reporting facilities, and its role binding.

Compared to Process Weaver the weak features of the PCMS*CTS environment is the lack of general chaining and notification mechanisms and the lack of automation capabilities. The state transition approach cannot handle concurrent threads of control. The consequence of missing automation capabilities is that the process programmer is strictly constrained by PCMS*CTS functionality. Most weak features can be related to the incapability to actively use the “affected” relationship. Process agents cannot select an “affected item” and then perform normal PCMS operations on the selection, e.g., extract item, action item, update item attributes.

To summarize, PCMS*CTS is strong on process management issues, but relatively weak on coordination and automation issues. In the following we identify missing functionality in PCMS*CTS.

#### 8.2.1 Multiple products

The most fundamental problem with PCMS*CTS is that it does not support changes that involve documents from two or more PCMS product structures, i.e., a change document belongs to a given PCMS product structure, and cannot refer to PCMS items in another PCMS product. Since change and review processes in the studied organization often involve several products, the PCMS*CTS implementation cannot be used.

#### 8.2.2 Update item attributes

It was needed to associate information such as editor, controller, and status to each of the affected items. Although items can be listed as affected, it is impossible to annotate these items with additional information. Thus, to annotate affected items one needs to use attributes defined on items. The problem is that attributes of affected items cannot be updated from a change document, each document must be found manually by using the “product browser”.

199
When found, the document is selected and the attributes updated using the “edit attributes”
function in the browser menu.

8.2.3 Allocate agents to affected items

Agents cannot be allocated to items listed as affected in a change document. The agent–item
allocation was characteristic for the two processes, i.e., each item should be updated by a
specific agent. Furthermore, this is likely to be a characteristic of any process, and should be
supported by the change document. The needed functionality is to be able to partition the
items according to responsible agents, and then let the agents control their partitions until
the update has been made.

8.2.4 Extract, return, or action affected items

A PCMS*CTS change document cannot action affected items to a new lifecycle state. The
change document itself can be actioned by agents who have the change document in their
pending list. The affected documents can be actioned by any PCMS user who has the appro-
priate access rights in the part hierarchy, i.e., this is neither supported or constrained by the
change document. There might be relationships between item lifecycles and change document
lifecycles, but this is related to lifecycle states and not PCMS users.

The needed functionality is to partition the affected items according to responsible agents.
Furthermore, the change document should offer functionality to be distributed to all respon-
sible agents, but with access rights according to the agent partition. The agent who controls
a partition should be able to extract, return, and action items in his partition.

8.3 Process Weaver experiences

All strong features of the Process Weaver environment can be related to the Petri-net formal-
ism, the customized work environment, and the ability to write general code in the cooperative
procedure. First, the parallel threads and synchronization mechanism of the Petri-net is more
powerful than the state transition approach offered by PCMS change documents. Second, with
the Process Weaver work contexts one can make simple, customized work environments for
each process step. This makes the work environments easy to understand and easy to work
with, as compared to change documents (i.e., information filtering and bypassing complex
change document dialogues). Third, the ability to write general code and invoke external
programs is what makes the cooperative procedure capable of doing useful work, e.g., access
the repository and repository information.

Most weak features of the Process Weaver environment is related to its lack of integration
with the repository. The consequences are that process and product information become
unrelated, it is difficult to handle constraints between processes and products, and there is no
uniform way to access product and process information. Furthermore, Process Weaver does
not offer process history and process reporting.
To summarize, although Process Weaver offers strong coordination and automation support, it is weak on process management issues. In the following we identify some missing functionality in Process Weaver.

8.3.1 The agenda

The only problem we have encountered with the agenda is that it becomes difficult to keep an overview of the arriving work-contexts. The Agenda should provide a way to automatically group work-contexts sent from a single Cooperative Procedure. Since a cooperative procedure may contain other cooperative procedures this should possibly be done in a hierarchical manner. As of today, the user must create his own categories and manually move arriving work-contexts into a given category.

8.3.2 The work-context

We find that Work Contexts offer too little functionality to the process programmer. The main deficiency discovered is that Work Contexts cannot have any associated executable code. In order to do useful work one is forced to return control to the cooperative procedure, do the work there, and then receive another work-context (selecting documents, error handling, etc.). The result is that tasks become fragmented and are not viewed as tasks by the performing agent. In particular, we find that the following should be improved:

- **CoShell code.** Although one may use variables within a Work Context one cannot associate CoShell code with the work-context. E.g., one cannot take action on a list selection. If the Work Contexts had been active we could have saved much more effort in the process of selecting documents from the PCMS repository. As of today, we had to write each selection into a file, and then use the cut and paste of the window manager to enter these values into a Work Context document list. With executable code this could have been automated by querying the repository, and updating a WC selection list with the results.

- **Variable number of documents.** The number of documents needed in a work-context is often determined at run-time. This cannot be handled by the work-contexts. The result is that one must send one WC for each document, although the documents are logically related. E.g., the documents to be updated by an agent after the review meeting, or the documents to be changed by an agent in the change management process.

- **Answer button.** There should be some validation code associated with an answer button, i.e., the work-context should not terminate until a condition was satisfied. As of today, this must be done in the procedure and the work-context must be sent back to the user. This clutters the Petri-net and is awkward for the user who must wait for new work-contexts to arrive. Logically, such validations belong to the work-context, not to the cooperative procedure. E.g., if the change manager does not select a change responsible in the Change Management process, he will receive the work-context once again. However, he might think he selected an agent, and continue with other work, leaving the process blocked until he discovers the mistake.
To conclude, the work-context editor should offer more functionality to the process programmer. We find that work-contexts should be general programs, not hard coded as in Process Weaver.

8.3.3 The cooperative procedure

Although a work-context can be sent to a user without involving a Cooperative Procedure, the cooperative procedure is considered as the most fine grained executable process in the Process Weaver environment, since the former corresponds to sending an e-mail. Furthermore, we find it useful to separate between a *process step* and a process fragment, a process step being a process fragment that cannot be further decomposed. A cooperative procedure then corresponds to a *process fragment*. Seen from the process participant a process step becomes a single work-context, from the point in time when it arrives on the agenda, until it disappears from the agenda. Seen in this way, a *process step* has no direct representation in a cooperative procedure. To model a process step we need two transitions linked by a place, one for sending a work-context, the other to wait for the user to finish his work. Therefore, we recommend that all user interactions are decomposed into a separate cooperative procedure, else it will be difficult to read the Petri-net, and it will be difficult to reuse user-interactions.

In general, when applying Petri-nets for the purpose of process modeling we have found a range of problems:

- There are no concepts in the Petri-net that correspond to an atomic work operation, i.e., it is a poor abstraction for work processes.

- The Petri-net metaphor easily breaks down because the programmer is not constrained by the Petri-net when defining the process logic (because of the global scope of a variable).

- The Petri-net is not capable of expressing certain situations. E.g., when two threads are merged by a logical OR, but then split again according to the thread that was executed.

- In other cases it might be possible to express the needed logic with a Petri-net, but it is not feasible because of the complexity (low readability) of the resulting Petri-net.

- There is no concept of process exceptions, exceptions must be modeled just like the normal flow of operation. The consequence is that the Petri-net becomes unnecessarily large and unreadable (often there are few normal threads, and a large number of exceptions). E.g., most work-contexts provide values to the sending procedure upon its termination. These values must be checked and the work-context resent upon errors which cause an extra link. Furthermore, the user should be noted about the failure, this requires one additional transition to send a different work-context.

At the large, we find that the Petri-net formalism is too low level for process modeling (the assembler of software process modeling). Some, but not all of these shortcomings can be overcome by using colored Petri-nets as in SPADE (Petri-nets with typed tokens).
8.4 Summary

This Chapter has discussed experiences with PCMS*CTS and Process Weaver. PCMS*CTS is found to be strong on process management issues, e.g., process control, process history, and process reporting. Most of this stems from a tight integration with the PCMS repository. PCMS*CTS is found to be weak on process coordination and automation issues. On the other hand, Process Weaver is strong on coordination and automation support, but weak on process management issues. Strong features of the Process Weaver environment can be related to the Petri-net formalism, the customized work environment, and the ability to write general code in the cooperative procedure.
Chapter 9

Evaluating overall questions

All the overall questions are posed within the context of software development processes performed within the boundaries of a project. Furthermore, it is assumed that a configuration management system is in use, and that every artifact produced by the project is stored as an SCM item.

The format of the following discussion is to repeat the question stated in Chapter 2, then to give a short answer followed by the argumentation.

9.1 Process modeling – actual process

**Q1:** Can existing modeling support (PML and PME) be used for industrial processes?

**Answer:** To the extent that industrial processes are sufficiently represented by type level models the answer is yes. However, it is not likely that these models are correct with respect to the actual development process, and thus, the modeling is of limited value.

Process modeling is a difficult activity, because of the intangible nature of the process concept. The problem that arises is that there is no agreement upon what is “really” the existing process. There exists a range of perspectives, e.g., as reported by [8]: Official process, perceived process, desired process, actual process, observed process. One of the fundamental assumptions of software process modeling is that these perspectives will be harmonized by providing an explicit and well defined model.

As of today most process technology assumes that there is, or should be, a one-one correspondence between the process handbook and the performed process. Essentially, experience papers on process modeling are concerned about modeling the generic method, assuming that this in itself will improve the development process. My thesis has identified three modeling levels, namely the generic model (GEH), the specific model (EHS), and the instance model. The instance model is represented by PCMS product structures, Change reports, and Review reports.

The real-world process: It is difficult to model the actual development because it is subject to uncertainties and exceptions. These are essential properties of the actual process which
will be present even when the modeling is done by process performers. This because process performers cannot have the process overview (perceived and observed), they mix the idealized processes (official and desired) the actual process. Furthermore, process instances are never performed identically, and two projects have different processes. Thus, as time goes by, process performers are forgetting how processes were performed, and they are mixing different projects and executions. My answer to this problem is to document the real-world processes in terms of instance level models. The motivation for this is the usefulness of the product structures found in PCMS in order to assess the real-world project processes.

Instance level models. There is a need for explicit instance level models which are created and maintained within a given software project. The instance level model should be as close to the actual development process as possible. With respect to process modeling, the instance level model should act as a definition of the actual development process. In the studied case this level corresponds to PCMS product structures. In general it will correspond to all kinds of project specific information, e.g., the project plan, calendar time, deadlines, named products, named human resources, etc. A named process participant responsible for a named document is an example of an instance level relationship.

Specific type level models. There is a need for type level models which are specific to individual projects. The specific model is an abstraction and generalization of the instance level model. The instance level models should be created from, related to, and give feedback to the project specific type level model. In the studied case this level corresponds to the Engineering Handbook and the PCMS control plan.

Generic type level models. There is a need for type level models which are generic to individual projects, but specific to the organizational context in which projects are executed. The generic model is a generalization of the project specific type level model. Project specific models should be created from, be related to, and give feedback to the generic model. In the studied case this level corresponds to the Generic Engineering Handbook.

Maintaining a generic description and a set of project specific descriptions causes a new set of requirements on model consistency. At the same time, project specific models must be allowed to be inconsistent with the generic model, and the generic model must be updated by experiences in individual projects. E.g., for process improvement purposes one always measures attributes of specific projects, but measurements are normally only compared with respect to the generic model. Thus, if relationships are not maintained one cannot state whether measurements are comparable over a set of projects.

Q2: Which are the benefits of using existing PML’s and PME’s?

Answer: Consistency and completeness of the process manual can be improved. However, with APEL the cost is too high, mainly because the language is too complex.

My work strengthens the belief that consistency and completeness of the process manual can be improved by the application of a formal modeling language. However, there is two conflicting requirements that must be balanced, namely expressibility and understandability. The formal model is only useful to the extent that it can be understood by personnel that cannot be expected to have training in the formalism, neither have the time or motivation for learning the formalism. This experience can be summarized as requirements on PML’s and PME’s.
• **PMLs.** Simple and easy to understand PML, few but relevant constructs, formally defined syntax and denotational semantics, simple diagrammatic language.

• **PMEs.** Powerful graphical support, powerful documentation support, and powerful correctness and consistency checking.

The APEL language and environment fails with respect to these requirements, mainly because the environment aims at producing enactable process models. To be enactable there is a need for a range of concepts which were irrelevant in my process modeling activity. Furthermore, other concepts that were relevant were not included, e.g., to differentiate between paper copies and configuration items.

**Q3: Is software process modeling cost effective?**

**Answer:** This question is still unanswered, mainly because the benefits of software process modeling are difficult to measure. However, there are some indicators showing that PME’s should focus on reducing modeling costs.

There are some indicators in the Company that maintaining process descriptions takes too much time and effort: In a period of two years the company used 60% of a person-year on GEH and EH creation and maintenance. In a single project 8% of the project effort were used on EH maintenance. This figure is judged to be too high by the Company, an acceptable level is about 1% of project effort. The goal is to reduce the effort needed for instantiation, maintenance, and feedback, while maintaining or increasing the EH quality. This is the Company’s motivation for looking into formal process modeling languages and supporting environments.

The main concern of the Company was to reduce effort, not to increase the EH quality. We evaluated it to be more costly to make models in the APEL environment than in a standard drawing tool (Idraw). The main reason is that there are too many constructs and graphical symbols in APEL, i.e., the modeling becomes difficult and slow. To be beneficial in the studied case there are four main issues that must be improved: Fewer and simpler constructs (e.g., remove the distinction between control and data flow), more power in the graphical interface (e.g. propagation of activity inputs and outputs when the activity is decomposed), consistency checking (e.g., decomposition tree, product model), and documentation support (e.g., such that APEL can replace the EH).

### 9.2 Process enactment – actual process

**Q4: Can existing enactment support (PPL and PE) be used for real world processes?**

**Answer:** The answer must be divided into single, individual process fragments, and composed process fragments:

• For single, individual process fragments that are well understood the answer is yes. This is demonstrated in the experiment with the change management and the document review processes.
• For general systems development processes the answer is no. These cannot be pro-
grammed because they cannot be strictly chained. There are too many unknowns such as:
the exact start and end criteria of process steps, the exact multiplicity or the in-
stantiation of process steps, the exact binding strategy between process instances and
between process instances and product instances, and the availability of resources.

Q5: Which are the benefits of using existing PPL’s and PE’s?

Answer: The traditionally stated PSEE benefits were not found to be important. Traditionally stated process enactment benefits can be divided into coordination, enforcement, automation, and guidance support. Process enactment systems have the potential for pro-
viding benefits which extend these “traditional” benefits. Such added value is based on the
knowledge of the actual development process, not only the specified process. What is likely to
be important are process monitoring and process history. This can be realized if enactment
systems become concerned with process documentation rather than process specification.

Issue: Are coordination, enforcement, automation, and guidance support important with
respect to the studied case?

• Process coordination. Benefits may arise from enabling people to be more effective, e.g.
chaining and notification.

The studied organization does not see process coordination support as an important
issue. That is, coordination support which extended the existing PCMS functionality,
i.e., access and concurrency control. The lack of interest in process coordination can
be explained by a relatively small group (15 people) located at a single floor in a single
building. The effect of a larger development group can be that processes are executed
more often. The effect of a distributed group can be that there are more to gain in each
execution.

• Process enforcement. Benefits may arise from process enforcement are conformance
benefits, i.e., it is ensured that the performed process is equal to the prescribed processes.
Process conformance is seen as an important issue from a quality assurance perspec-
tive. The problem is that processes cannot, in general, be enforced by means of process
programs. This because there are too many uncertainties and exceptions in the develop-
ment process.

• Process automation. Benefits may arise from work being done by the machine, rather
than by people.

The experiment shows that effort can be reduced, but not to an extent that the invest-
ment in Process Weaver is worthwhile.

• Process guidance. Benefits may arise from the system having more process knowledge
than a given process agent. Systems inspired by artificial intelligence and expert systems
aim to support process guidance. Support in this category includes help functionality,
decision support, and general information about the process.

Such benefits are not addressed by this work. In the studied organization the process
knowledge was either documented in the Engineering Handbook, or solely in the minds
of process agents.
Process enactment systems have the potential for providing benefits apart from the coordination and guidance benefits. This has been exemplified with the PCMS*CTS environment which is weak on both coordination and automation, but strong on process control, process history, and process reporting. Although the experiment shows that the PCMS*CTS process implementations not save any effort, it was judged to be a better solution by the Company. This because it was a low cost implementation with added value, due to the need to know the actual development process, not only the specified process.

- **Process management.** The need for process control, process history, and process reporting. Benefits that arise from increased availability of structured, high-quality, up-to-date information.

- **Process modeling.** If the actual development processes are known the process modeler may create generic models from specific cases in an inductive way. As of today, the process modeler must create abstract models without knowing the specific instances. He may create type level models on the basis of existing instance level models.

- **Quality assurance.** The need for comparing the actual development process with the specified development process. If both are modeled, there is a possibility for conformance reports.

- **Process improvement.** The need for data gathering and analysis for the purpose of empirical assessment of process properties.

The organization saw immediate benefits from PCMS*CTS, the only reason for not applying the PCMS*CTS module is that it does not support multiple products (change documents cannot reference items outside the product in which it is defined).

**Q6:** Is process enactment cost effective?

**Answer:** Within the processes of the studied case, with Process Weaver as the PSEE, and PCMS as the SCM system, I find the answer to be no, for the following two reasons:

1. *There are too few processes that are enactable.* The direction for solving this problem is to introduce instance level models, and to introduce other chaining strategies than the finish–start dependency. This would make more processes enactable because more information is known at the time when instance level models are created. The finish–start constraint is simply unrealistic.

2. *Enactable processes are executed too few times to be cost effective.* The direction for solving this problem is to reduce process programming costs, or to provide added value to project stakeholders. It is unlikely that these properties can be achieved without a close integration between the applied PSEE and the applied SCM system. In the experiment most of the implementation cost was associated to performing PCMS operations, and the added value achieved by PCMS*CTS is based on the SCM integration.

**Q7:** Can existing instantiation support be used for real world processes?

**Answer:** No, because of the lack of instance level models.
The instance level must be explicitly modeled, not automatically generated from the type level description. The latter is unrealistic, e.g., a project manager would not accept that her project plan was deduced and manipulated by a computer program. This is a sub-problem to the lack of instance level models, and PSEE’s should at least demonstrate that they can handle this problem before claiming that human intervention is not needed.

9.3 Process modeling – process enactment

**Q8: Can every modeled process be enacted?**

**Answer:** No, because real-world documents and people can be modeled but not programmed.

If we model the processes as they are performed in an organization, we find that many processes are not programmable, e.g., a project meeting, filling in a paper form, sending paper copies, checking the consistency between paper copies and repository items, creative work, phone calls, etc. SCM and SPE systems are only concerned about products and processes that can be handled by computer programs. Process modeling in general is concerned about all aspects of the software development process.

Automation boundaries in the studied case: In software engineering work much of the information exists as documents distributed in the form of paper copies. Such paper copies cannot be handled by PM and CM systems, although they may be essential to the development process.

- Information about the development process is lost if we only consider CM items (e.g., we miss the essential parts of the process).
- If we consider both paper copies and CM items, we get modeling problems with respect to identities. E.g., to model the relationships between paper copies and the stored CM items.

**Q9: Can every programmed process be enacted?**

**Answer:** No, because the finish–start constraint can always be programmed, but often not enacted.

Process enactment systems with strict chaining, as exemplified by Process Weaver’s cooperative procedures, will only be able to enact fragments of the overall system development process. The reason for this is that the overall development process is subject to too many unknowns, uncertainties, and exceptions. Strict chaining can be programmed but not expected to be followed in the actual process.

Systems development processes cannot be programmed, since Process Weaver only provides strict chaining, and these processes are usually overlapping. What are the properties of processes which can both be programmed and enacted. This thesis identified two enactable process fragments within the studied case, namely Change management and Document review.

**Q10: Should the same language and environment be used for both modeling and enactment?**
**Answer:** No, because they have different purposes and solves different problems.

**Different purpose:** The purpose of a process model is different from the purpose of a process program. A process model is a tool for process understanding and definition, while a process program is a tool for computerized support of some part of the process. The consequence is that PMLs and PMEs are subject to requirements which are different from PPLs and PSEEs. These differences are so huge that it is not advisable that the same language should be used for both purposes.

**Different problem:** The problem of process understanding and definition is different from the problem of computerized process support. Even when a single language is used for both modeling and enactment, the created model is not necessarily enactable, and furthermore, should not be enactable. A process model should not be concerned about enactability, because this disturbs the main issue, namely understandability. The language does not decide whether a specification is enactable or not, this is decided by the problem which is solved. A process model can only be refined into a process program if the goal, right from the start, is to create process programs. But then one is not concerned about process modeling, but about process programming.

### 9.4 Process modeling – SCM

**Q11:** Which are the relationships, if any, between process modeling and SCM?

**Answer:** The SCM system has a product model, this must be harmonized with the SPM product model, or a mapping must be provided.

It is of little use to have an advanced PM product model which cannot be supported by the existing CM system. The two models should not be different, if they are different there must exist a mapping between the concept and its repository manifestation. Product lifecycles must be chosen on the basis of existing company processes, and must involve a strategy for creating new item revisions. Lifecycle design must consider the total set of activity types and product types because a product type can be used by many activity types, and an activity may use many product types. The choice of product abstractions should be made according to strategic goals, and according to needs of different categories of personnel, e.g. project participants, project management, process modelers, quality assurance personnel, and process improvement personnel.

### 9.5 Process enactment – SCM

Most PSEEs do not provide SCM functionality. From this fact one should expect one of the following to be true: The PSEE can be used together with any SCM system, or SCM systems are not needed when a PSEE is applied. If both statements are false we can conclude that a PSEE without SCM functionality cannot be applied to software engineering projects. Furthermore, that the only viable strategy for a PSEE intended for software projects is to be integrated with an SCM system.
Q12: Can PSEEs replace SCM systems?

Answer: PSEE as exemplified by Process Weaver cannot replace SCM systems as exemplified by PCMS. On the other hand, PCMS is needed by the studied organization, Process Weaver or not.

Q13: Can PSEE be used together with any SCM system?

Answer: PSEE as exemplified by Process Weaver can be used together with a SCM system as exemplified by PCMS. However, as shown by the experiment, this approach is not cost effective.

The integration is expensive to implement and provides neglectable benefits to the organization. The two main reasons are:

1. **Model consistency.** In cases where independently developed PSEE systems and SCM systems must be used, one should expect problems associated with differences in the product model. In the case of Process Weaver there is no product model, everything must be handled by the process programmer.

2. **The process layer.** A PSEE introduces a new conceptual level between the repository and the user, namely the process layer. E.g., as in SPADE, the process layer can be seen as a routing of events between tools and users. In this respect there is a conceptual mismatch between SCMs and PSEE systems, because SCMs rely upon user identities, while the PSEE relies upon process identities. In the case of the Process Weaver and PCMS integration every PCMS “extract” and “return” was made by the “weaver” user, leaving no information about processes or users in the PCMS repository.

If PSEE and SCM systems are separately developed and not harmonized with respect to provided functionality, the usability of process enactment is significantly reduced. It is reduced because there is functionality which is needed but cannot be implemented, and because the functionality that can be implemented comes at an unreasonable implementation cost.

Based on the studied case and the experiment one can reject the “replacement hypothesis” and the “loosely integrated hypothesis” and state that PSEE and SCM systems must be integrated to be useful to software engineering projects. The consequence is to ask how they should be integrated.

Q14: What are the requirements for an integrated PSEE and SCM system?

Answer: The experiences with Process Weaver and PCMS makes me believe that the following two properties are essential to PSEE and SCM integration:

1. **Common product model.** An integrated PSEE/SCM system must have a single product model, both with respect to types and instances. The concept common to both PSEE and SCM systems is the product concept. Although the modeling concepts might be different in style and scope, a model of the product is always present in both system categories. A system that does not integrate these aspects is not integrated.

2. **Encapsulated repository.** An integrated PSEE/SCM system must completely encapsulate the repository by means of processes. If the SCM repository interaction and content
cannot be controlled by the PSEE system, the process layer is of little value to software projects. This because one cannot make assertions about the relationship between processes and products, i.e., one cannot maintain producers and consumers relationships. A system which allows users to bypass the process layer is not an integrated system.

To summarize, PSEEs must be closely integrated with SCM systems to be of any use to software engineering projects. The main properties of an integrated system is a common product model, and an encapsulated repository.
Chapter 10

Conclusion

The overall goal of this thesis is to provide experiences in software process modeling and enactment. Such experiences are reported as modeling experiences in Chapter 5 and enactment experiences in Chapter 8. Furthermore, the introduction states a set of overall questions which are discussed and evaluated in Chapter 9. This Chapter states the main result, describes related work, and presents a vision for the future.

10.1 Main results

This section states the main results according to thesis parts, namely process modeling and process enactment.

10.1.1 Process modeling

With respect to process modeling my main result is the identification of instance level models, and the role of instance level models as a representation of actual development processes.

As of today most process technology assumes that there is, or should be, a one-one correspondence between the process handbook and the performed process. Essentially, experience papers on process modeling are concerned about modeling the generic method, assuming that this in itself will improve the development process. My thesis has identified three modeling levels, namely the generic model, the specific model, and the instance model:

1. *Generic type level models.* There is a need for type level models which are generic to individual projects, but specific to the organizational context in which projects are executed.

2. *Specific type level models.* There is a need for type level models which are specific to individual projects.

3. *Instance level models.* There is a need for explicit instance level models which are created and maintained within a given software project.
The instance level model should be as close to the actual development process as possible. With respect to process modeling, the instance level model should act as a definition of the actual development process.

10.1.2 Process enactment

With respect to process enactment my main result is the experimental result with respect to the cost effectiveness of process process programming.

Within the processes of the studied case, with Process Weaver as the PSEE, and PCMS as the SCM system, I find process programming not to be cost effective, for the following two reasons:

1. **There are too few processes that can be programmed.** The direction for solving this problem is to introduce instance level models, and to introduce more flexible chaining strategies.

2. **Those that can be programmed are executed too few times to be cost effective.** The direction for solving this problem is to reduce process programming costs, or to provide added value to project stake-holders.

It is unlikely that a solution can be achieved without a close integration between the applied PSEE and the applied SCM system.

Although I believe that it can be generalized and valid for most software engineering projects, this is not proved by the studied case and the performed experiment. However it can be justified in the following way:

1. **There are too few processes that can be automated.** A high degree of automation is expected to be strongly correlated to well defined processes, a stable problem domain, high maturity level, and strong certification requirements from customers. These properties are fulfilled by the Company, but generally not fulfilled by the software industry. Thus, I expect the Company to have higher than average possibilities for process automation.

2. **Those that can be automated are executed too few times to be cost effective.** In the studied case the automated processes are executed many times, estimated to 300 times a year for the change management process and 250 times a year for the document review process. Even if the size of the development group is only 15 people, I do not expect that larger organizations will execute processes more often. The size of the organization is irrelevant if they cannot execute the same process program. Problems and tools, thus the processes, are different between software development groups, even when they are part of the same organization. At least, before using the size of the organization as an argument one should justify that they can execute the same process program in different groups or departments.

The studied organization was located at a single floor in a single building. In a distributed group there may be more to gain in each process execution. This is not addressed in my work.
10.2 Related work

My case study approach is more explicit than the ones reported by others, e.g., the case and involved personnel is more clearly defined. Other case studies are performed by means of "own" languages and tools, while I have used languages and tools developed by others. Furthermore, it is the first case study which uses information from actual development projects to assess process models.

My experimental approach to assessing the benefits of process enactment is rather novel. To my knowledge this has not previously been done. Here I divide process technology into process modeling and process enactment technology, and briefly mention how my work is related to the published experiences in these categories.

10.2.1 Process modeling

By process modeling experiences we mean the application of formal or semi-formal process modeling languages for the purpose of describing software engineering processes. To our knowledge there is only one paper that describes an experiment\(^1\), namely Votta [118], in which an experiment that compares one formal and one informal process description is reported. The major conclusion is that the formal process description was more succinct, accurate, and complete, and as a consequence easier to understand, integrate, and measure. Votta states, however, that the return of investment is still an open question.

[3, 4, 8, 80, 84] report experiences from the formalization of handbooks and quality manuals. The results are similar to those reported by Votta, but these publications do not perform experiments, only loosely defined and informal case studies. However, they do not investigate the relationship between the handbooks and the actual process, and they are not concerned about the role of configuration management systems. These works do not identify the need for instance level modeling, or the difference between project generic and project specific type level descriptions.

McGowan and Bohner [91] describe Model Based Process Assessments (MBPA), an approach used to combine process modeling with process assessment. The major point in this paper is that a diagrammatic (IDEF0) representation of the software processes facilitates process understanding and communication among the personnel involved in the assessment. The paper does not describe an experiment, but contributes by describing how models are developed and used. Otherwise, their work is similar to those mentioned above.

10.2.2 Process enactment

Bronisz et al. [20] describe the introduction and use of Process Weaver internally at Cap Gemini Innovation, with a process enacted in a real-world environment. My work reports processes enacted in a laboratory environment. Two things are common to these processes, both use Process Weaver, and both enact process fragments. My work is different in that

\(^1\)Votta states it to be an experiment, but from the available description it seems more like a quantitative case study as defined in [83].
it concerns software development processes, it applies a configuration management system (PCMS), performs a controlled experiment, and discusses economical feasibility. Furthermore, I discuss the extension from process fragments into a project context in the form of how fragments can interact through the configuration management system.

Barghouti et al. [9] report experiences from two case studies at AT&T applying Marvel as a process modeling environment. Here, AT&T sample data were used to verify the models by executing scenarios, simulating the actual execution of the processes. The latter is similar to what we report in this work, with the difference that Barghouti et. al. did not perform an experiment. Furthermore, Barghouti et al. do not use a configuration management system, and they do not discuss the interaction between process fragments.

10.3 Future work

There is a substantial cost of applying both process modeling and process enactment to software projects. As of today, process technology provides too few benefits compared to the cost of its application. Moreover, I find that it is hard to significantly reduce its cost, because the main problem is to know the process and not to describe it. It is difficult to know project processes because of uncertainty and instability, it is hard to know both what the processes are, and how they should be.

Process technology has many promising possibilities, but that its potential has not materialized in existing environments. To materialize, we find that the focus should be less on process automation and more on process tracking. I.e., that strict chaining is unrealistic for all but trivial processes. We find that the two most promising directions are process traceability and soft enactment.

10.3.1 Process traceability

*Process traceability seems to be the most promising potential advantage of process technology.*

Based on the case study I identify that improved *internal quality* (verification) of the process description is not a major problem, and even the problems that exist are not likely to be solved by process modeling in a cost effective way. The most prominent problem is *external quality* (validation), to ensure that the description is in correspondence with reality.

After evaluating potential benefits of process technology against the studied projects, we concluded that the most promising of the stated benefits is the possibility for a cost effective *process history*. That is, if a process history can be established, a software project can benefit from this in numerous ways, e.g., with respect to project management, quality assurance, and process improvement. Moreover, the external quality of the process description can be assessed, the model improved on the basis of empirical data.

10.3.2 Soft enactment

*Software projects can be enacted in a soft way – chaining cannot be enforced.*
The problem of process traceability has not been addressed sufficiently by the research community, e.g. based on our experiences with Process Weaver and repository integration, we conclude that Process Weaver would not provide the needed functionality.

The vision for the future is to encapsulate the repository with a task layer, i.e. nothing can be extracted from or returned to the repository without a task-identity. Tasks are instantiated from a hierarchical set of activities. The individual work environments are asynchronous and may at any time acquire inputs and produce outputs. The only constraint is that a task-identity must be provided, an individual may violate any other rule that might exist (chaining, typing, pre-conditions, etc.). The system stores information about all tasks in a way that is explicitly accessible, i.e., the process trace.
Bibliography


221


