Software Architecture and SCM,
INCO project and component-based development

Bernhard Westfechtel and Reidar Conradi
RWTH, Aachen, Germany /
NTNU, Trondheim, Norway

Extended for IFIP WG 2.4 meeting on „Software Components“,
San Miniato (Pisa), 28.5-1.6 2001
Software process (simplified view)

- requirements engineering
- programming-in-the-large
- programming-in-the-small
Definitions

An **architecture** is the set of significant decisions about the organization of a software system, the selection of the structural elements and their interfaces by which the system is composed, together with their behavior as specified in the collaborations among those elements, the composition of these structural and behavioral elements into progressively larger subsystems, and the architectural style that guides this organization.

[Booch, Rumbaugh, Jacobson 1999]

I.e. system - components - interactions ("connectors")

**Configuration management** is the process of identifying and defining the items in the system, controlling the changes to these items throughout their life cycle, recording and reporting the status of items and change requests, and verifying the completeness and correctness of items.

[IEEE 1983]

*To control the evolution of large and complex (software) systems.*

[Tichy 1988]
Questions/goals

Investigate the borderline between software architecture and SCM
  » Product space
  » Version space

Investigate the division of labor between ADL tools and SCM systems
  » To what extent may an ADL tool rely on services provided by an SCM system?
  » To what extent should an SCM system already incorporate support for software architecture?
  » From the perspective of an ADL tool builder:
    What are the requirements to SCM services?
    In what respects do current SCM systems not (or not adequately) address these requirements?
## Architectural Description Languages (ADLs)

<table>
<thead>
<tr>
<th>Architecture Modeling Features</th>
<th>Tool Support</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Components</strong></td>
<td>Active Specification (GUI)</td>
</tr>
<tr>
<td>» Interface, Types, Semantics,</td>
<td>Multiple Views</td>
</tr>
<tr>
<td>Constraints, <em>Evolution</em></td>
<td>Analysis</td>
</tr>
<tr>
<td><strong>Connectors</strong></td>
<td>Refinement</td>
</tr>
<tr>
<td>» Interface, Types, Semantics,</td>
<td>Code Generation</td>
</tr>
<tr>
<td>Constraints, <em>Evolution</em></td>
<td>Dynamism/Evolution</td>
</tr>
<tr>
<td><strong>Architectural Configurations</strong></td>
<td></td>
</tr>
<tr>
<td>» Understandability,</td>
<td></td>
</tr>
<tr>
<td>Compositionality,</td>
<td></td>
</tr>
<tr>
<td>Heterogeneity, Constraints,</td>
<td></td>
</tr>
<tr>
<td>Refinement and Traceability,</td>
<td></td>
</tr>
<tr>
<td>Scalability, <em>Evolution</em>,</td>
<td></td>
</tr>
<tr>
<td>Dynamism</td>
<td></td>
</tr>
</tbody>
</table>

[Medvidovic and Taylor 1997]

- Component
- Connector
- Configuration

**Representations**

**Properties**
- shape = rect.
  width = 100
  height = 50
  color = blue
- Visualization Spec.
  ... while(data)
  read(response);
  ...
- Source Code
- Throughput = 5 kbps
  max_connect = 10
  ...
- Performance Data

**Representation Variant**
- Small-memory Representation
  Small-mem-RM : RepMap

**Representation Variant**
- High-Performance Representation
  High-Perf-RM : RepMap
ADL Examples (3): Wright [Allan and Garlan 1997]

```
component Client =
  port Request = ...

component Server =
  port Provide = ...

connector Service =
  role Client = request!x → result?y → Client ∏ √
  role Server = invoke?x → return!y → Server √
  glue = Client.request?x → Server.invoke!x
        → Server.return?y → Client.result!y → glue √
```
**ADL Examples (4a): Ménage [van der Hoek et al. 1997]**

```plaintext
InterfaceExemplar
name = Point;
revision = 2;
representation = {
    integer x;
}

ComponentExemplar
name = GlobalOptimization;
revision = 3;
interfaces = result;
components = opt, eval;
connections = bus1, pipe1;
...
ascendant = (none);
descendant = ComplexGlobalOptimization(1);

Interface
name = result
direction = in
interfaceExemplar = Point(2);

ComponentExemplar
name = ComplexGlobalOptimization;
revision = 1;
interfaces = result;
components = complexOpt, eval;
connections = fastBus1, pipe1;
...
ascendant = GlobalOptimization(3);
descendant = (none);
```
ADL Examples (4b): Ménage [van der Hoek et al. 1997]

ComponentExemplar
name = GlobalOptimization;
revision = 3;
interfaces = result;
components = opt, eval,
       stat(statistics == true);
connections = bus1, pipe1,
     bus2(statistics == true);
properties = {
    statistics = true;
    speed = fast;
}

VariantComponentExemplar
name = Optimizer;
revision = 1;
interfaces = stat(statistics == true);
optionalProperty = speed;
components = slowOpt(slow),
       fastOpt(fast);

variant

variant

optional component
Software configuration management (SCM)

Versioned items: **software objects** and their **relationships**, w/ object types: source/derived, atomic/composite etc.

Composition of these items into **configurations**

**Version control:** evolution of above items, and generation of configurations=composites.

Use of a software repository and workspaces, w/ transactions

Representation of software architectures:
  » **Make files** (!)
  » **System models** dealing with
design objects (in ADLs, UML or SDL)
arbitrary software objects from all phases (in own SMLs)
System models for design objects (1): Adele I [Estublier 1984]

- **manual** m2_i2_v2
- **selimp**
  - m4-i2-v1.05
- **selcond**
  - *(syst = unix)*
  - *(connect = pipe)*
- **seldef**
  - *(type = debug)*
- **attributes**
  - type = debug
  - syst = unix
  - author = Jim
  - state = experimental
  - date = 84_05_13
- **end**

*built-in system modeling language*
System models for design objects (2): Adele II [Estublier 1994]

```plaintext
type Interface is Object;
  common
    system : {unix, hp, vms};
    graphics : {x11, open_win};
  modifiable
    belong-to : Conf;
    bug-reports : set_of Document
  immutable
    header : File;
    realization : versioned
Realization;
end;
```

user-defined versioned object type
System models for arbitrary software objects:

PCL [Tryggeseth et al. 1995]

family foo
  attributes os : \{Unix, VMS, DOS\};
  ...
  end
  parts
    root ⇒ main;
    files ⇒
      if os = Unix then unix_fs
      elsif os = VMS then vms_fs
      else dos_fs;
      ...
    end
  end
end
## Comparison of software architecture and SCM

<table>
<thead>
<tr>
<th></th>
<th>software architecture</th>
<th>SCM</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>coverage</strong></td>
<td>programming-in-the-large (high-level design)</td>
<td>whole lifecycle</td>
</tr>
<tr>
<td><strong>software objects</strong></td>
<td>components, modules, classes, ...</td>
<td>all kinds of software objects</td>
</tr>
<tr>
<td><strong>relationships</strong></td>
<td>inheritance, import, port connections, ...</td>
<td>all kinds of relationships</td>
</tr>
<tr>
<td><strong>semantic level</strong></td>
<td>high (syntactic or semantic consistency)</td>
<td>fairly low</td>
</tr>
<tr>
<td><strong>granularity</strong></td>
<td>overall organization, but also detailed interface descriptions</td>
<td>overall organization (software objects as black boxes)</td>
</tr>
<tr>
<td><strong>versions</strong></td>
<td>variants</td>
<td>variants and revisions</td>
</tr>
<tr>
<td><strong>version selection</strong></td>
<td>usually hard-wired</td>
<td>rule-based</td>
</tr>
</tbody>
</table>
Software architecture and SCM overlap since they both deal with the overall organization of a software system. However, while software architecture refers to programming-in-the-large, SCM addresses the whole software lifecycle.
The relation between software architecture and SCM can be studied under different assumptions concerning tool integration. It makes a big difference whether we assume bottom-up or top-down integration.

![Diagram](image-url)
Since there are many architectural description languages with different syntax and semantics, an SCM system must not prescribe a specific language for describing software architectures. In contrast, the software architecture description has to be handled like any other document.

Ex. Makefiles, system descriptions/designs in UML/SDL or ADLs, system models in Adele or DSEE, etc. Cf. also the model view/role problem.

Jacky Estublier, 2001:
- Advantages of SCM: *no* PL and related semantics assumed
- Disadvantages of SCM: *no* PL etc. assumed!
Under the constraints of bottom-up integration, there is no hope to unify software architectures and software configurations in the sense that only a single, integrated description needs to be maintained. A certain amount of redundancy is therefore inevitable.

Ex. (again) Make, UML/SDL, ADLs, SMLs (in Adele, DSEE, ...).
Architectural design deals with architectural evolution at a semantic level. In particular, variants can be represented within the software architecture description (as realization variants, subclasses, or instances of generic modules). In the way, planned evolution may be taken care of.

NB: “Planned” may mean either variants and/or revisions.
SCM offers general version control services for all kinds of applications. These address not only variants, but also revisions. Version control is performed at a fairly low semantic level. The SCM system deals with versioning of the architecture, covering in particular unplanned evolution.

Ex. Introducing another natural language in UI dialogues: can be done in many ways. First variant is always hard and not always foreseeable.

*Ralph Johnson: Reusable components are discovered, not planned.*
Altogether, the evolution of software architectures has to be dealt with at two levels: “structured” version control within the architecture (planned changes) and “unstructured” version control of the architecture (unplanned changes). These two levels complement each other and cannot be unified easily.

Again: “planned” does not imply variants only.

Architecture/system model may itself have revisions and variants (structural versioning, natural growth).
Summary: Spectrum of modeling alternatives

**Standard design languages**, such as SDL, UML etc., design and partly code, often no DB/UI stuff.

**Make files** (composition, dependencies, derivation, defaults): implementation objects, can be partly extracted.

**System modelling languages (SMLs)** from SCM systems: DSEE, Adele, EPOS-PDL, PCL, Clearcase etc., general objects/relationships, version support, commercial.

**Richer architectural languages (ADLs)**: Acme, Ménage etc., design objects/relationships, rich semantics, still research.

Need **traceability** towards rqmts/code, i.e. more than design.

*Reminder*: system models are costly to establish and maintain. Tool must offer useful support (generation, analysis etc.), otherwise the model will rotten and die.

Often just ”cloning” of a (sub)configuration in a shared directory, plus local \(\Delta\)'s. Subconfigurations as work breakdowns.
**Interplay between software architecture and SCM**

Redundancy cannot be avoided:
- How to keep consistency between architectural configurations and SCM configurations?
- How to select the appropriate level of granularity?

Potential problems of current SCM systems from the perspective of ADL tool builders:
- SCM version model does not match the needs of an ADL tool
- SCM version control is performed at a low semantic level
- SCM system enforces its own ADL
- Standardization issues in general

Desired features of SCM systems:
- Customizable support for representing design objects and their relationships
- More flexible version model
- Raise the semantic level of version and configuration control
INCO: Incremental and Component-Based Development

Newly accepted Norwegian basic R&D project, 2001-2004. Univ.Oslo and NTNU. 7 MNOK.
» Dag Sjøberg et al., Univ. Oslo (coord.).
» Reidar Conradi et al., NTNU.
» 3 PhD-students: Helge Koren, Parastoo Mohagheghi, NN.
» One postdoc: Marco Torchiano.

Need revolution in development paradigms -- waterfall is dead, “internet” time, extend RUP and similar models.
  Rqmts <=> design: a perpetual negotiation!

Component-based development, using OO / RUP / Commercial-Off-The-Shelf (COTS) -- but how to manage the risks?

Empirical studies on Norwegian IT industry and own students

Link to new Simula Research Lab in Oslo and PROFIT project.

Strong industrial and international cooperation.
INCO motivation (1): problematic requirements


- 31% dead-before-delivery, 81 bill.$ loss/year, 1% of GNP!
- 53% have serious overruns (189% average), 59 bill.$/year

Ex. Failed “long-duration” projects:
- TRESS, Rikstrygdeverket, 1989-1994, > 1,2 bill. NOK lost -- floating rqmts, but fixed budget and price.

Ex. Ericsson priorities per 2000:
- Faster, better, cheaper -- in that order.

Ex. Some “dot.com” projects:
- 2-4 weeks, e.g. in TV2, Infostream, and ICON Medialab.
**INCO motivation (2): Barry Boehm’s COCOMO-II**

Incremental development using COTS. Initial, iterative steps e.g. to scale and refocus project. Market-value driven.

<table>
<thead>
<tr>
<th>Traditional situation:</th>
<th>Current trends:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clear separation between requirements and design</td>
<td>Everything connected: e.g. available components influence requirements</td>
</tr>
<tr>
<td>Stable requirements</td>
<td>Rapid requirements change (50% change per 3 year)</td>
</tr>
<tr>
<td>Mostly in-house and controllable parts</td>
<td>No control over COTS evolution</td>
</tr>
<tr>
<td>&gt;2 year projects</td>
<td>Ever-decreasing lead times</td>
</tr>
<tr>
<td>Repeatable, mature models</td>
<td>Adaptable process models</td>
</tr>
</tbody>
</table>
**INCO: What is incremental development, general (1)?**

Main issue: **reduce the risks** with long development times, so get early prototypes/deliveries. Time-boxing.

Ex. Annotate and try out **RUP**.

E.g. in new **Genova** method to specialize RUP for web development by Genera sw house, w/ Univ. Oslo and PROFIT support. (Genera is Rational’s first European coop. partner).

Ex. Adapt and try out **DSDM** (Dynamic System Development Method), e.g. with Computas.

Also **HP Fusion, Catalysis, Spiral** -- all old, **Extreme Programming** as practised by Microsoft.
Some characteristics of incremental development [Karlsson2000]:

» What method (RUP etc.) used -- followed “as is”? 
» Length of increment, can it be dynamic? 
» Persons per increment, varying during increment? 
» Number of increments per project? 
» Timeframe of entire project? 
» Number of phases (analysis, design, …) per increment? 
» Data collection to improve development process? 
» ...

Empirical analysis in student project: 8 companies interviewed.
**INCO: Incremental development, challenges (3)?**

**Some open issues:**
- Length of increment, # phases involved?
- How to manage time-to-market vs. traditional qualities? How to manage the risks in general?
- Estimation models for this, based on Use Cases?
- What projects are most well-suited for this?
- What training is required from developers and managers?
- Try traceable requirements (e.g. DOORS tool) + reuse by COTS. Complicated process!
- What of this works in practice? -- need studies!
- ...
INCO: Component-based development, general (1)?

Ex. **OO frameworks** (“coupled” libraries with preplanned customization), e.g. BRIX framework from Veritas Software or Parlay for telecom.

Ex. Domain-specific **patterns**, as abstract modules.

Ex. **Software architectures**, as **product lines**.

Ex. NASA from 25% to 80% reuse in 1985-1993: 2X productivity, defect rates to 1/4. Biggest single gain of any new technology.

Ex. **REBOOT** project [Karlsson95] [PaciI97]: 30-50% extra cost in first project, 20-30% gain in later projects + savings in maintenance. => **Break-even after two reuses** (3 applic.).

Ex. Similar experiences from Ericsson Radar, Hewlett-Packard, Bull, Philips, Nokia, … and same for use of product lines.

Ex. Special **broker web sites** to trade components: www.componentsource.com (CBD), www.sourceforge.com (open system development). Also Objectware company in Norway.
Large potential to save costs and time, increase traditional quality, standardize architectures, on all kinds of artifacts, in many processes/projects, ...

But no easy task, i.e. no silver bullet:
» Component repositories “either empty or filled with rubbish” [Poulin1995].
» “Reusable components are discovered, not planned” [Johnson1995].
» Difficult architectural issues: design for future reuse/COTS, otherwise “plug and pray”.
» Difficult coordination across product lines and projects.
» Investing now for an uncertain future -- like for SPI, QA, and metrics programs.
So cannot “afford it” or it is cancelled, but software = capital!
INCO: Component-based development, COTS (3)?

Before: isolated domain-specific products, then reusable components across internal products, finally product lines. Now: COTS is eating its way into all this.

COTS problems:

» No comprehensive and flexible reuse process for COTS: e.g. how to evaluate/decide what when?
» Relation to incremental development and time-to-market?
» Must consider costs, schedules, risks, architectural issues, legal and economic issues, future potential.
» Some models on component characterization, others on cost/risk estimation, yet others on process aspects.
» Glueware issues: architectural styles + glue (wrappers etc.).
» How to express trust and future expectations?

Work e.g. at USC, UMD, SEI, … and at Philips, Siemens, …

Lots of work to do! – e.g. postdoc Marco Torchiano will start working on COTS characteristics.
NERA, Norwegian Telecom company: satellite telephony and digital TV, broadband radio nets. 200 developers in Oslo and Bergen.

Has 40 gateway/link stations worldwide against geostationary satellites.

Until recently, such a gateway has proprietary hw, OS, DB, switching protocols etc.

Now increasing COTS use, and internal “gateway components” project.

Ex. NERA checklist for COTS risks:
- Will missing functionality be delivered?
- Will promised functionality be delayed?
- Delivered with inadequate documentation?
- Extra training needed?
- What, if any, control of future evolution?

Case study under way, first as a prestudy to document architectures, components, and processes in summer 2001.

Later: e.g. make revised estimation models for INCO topics wrt. risks/costs – upstarting work by PhD student Espen Koren.
**INCO: Component-based development, Ericsson (5)?**

**Ericsson**, two development labs in Norway, mobile telephony, Internet telephony, AXE switches. 800 developers.

Ericsson, Grimstad: 500 developers, working on GPRS system (extension of GSM with packed-switched link stations, up to 250 Kbaud).

Ericsson, Gothenburg: 1000? developers, working on UMTS (a GSM successor, up to 2 Mbaud).

Platform: Erlang (realtime Lisp), C/C++, SDL/UML and TIMe method, DOORS tool?

Reusable components between these two projects, 60% reuse, 9 months negotiations to agree upon this.

Shared middleware and business layer, some COTS (next page).

Also reusable processes with standards, checklists etc.

Look especially at quality aspects (dependability) and processes.
Ericsson GPRS architecture:

Prestudy to document and analyze architectures, components and processes. Use same characterization framework as for NERA?

PhD student Parastoo Mohagheghi.
SPIQ/PROFIT: **better software quality for Norwegian IT industry**

NFR industrial R&D project. Jointly with Univ. Oslo and SINTEF in 1997-99, 2000-02. 3 PhD students, 4-6 researchers. 10 active companies.

Lead by Bravida Geomatikk (Telenor), attn/Tor Ulsund.

How to help smaller companies to improve?
Need insight from organizational sciences!

Pilot projects in companies, over 20 such.

Empirical studies, experience bases.


Emphasis: SPI under uncertainty, knowledge management, and studying novel software technologies.
SFF-Fornebu: Simula Research Lab, a national Center of Excellence (SFF = Senter for Fremragende Forskning)

A SFF in Software Engineering proposed by Ifi/Univ.Oslo (Dag Sjøberg) and IDI/NTNU (Reidar Conradi), July 1999


SFF-Fornebu => Simula Research Lab, Feb. 2001, four parts:
» Software Engineering (prof. Dag Sjøberg, coord.)
» Communication Technology (prof. Frank Eliassen, coord.)
» Numerical simulation (prof. Aslak Tveito, coord.)
» Undecided theme

Prof. Morten Dæhlen, Univ.Oslo as general director.

Decentralized: Fornebu, Oslo, Trondheim.

Total budget of 45 mill. NOK per year, 10 for each part.

Totally 25 teachers, researchers and PhD students; 30-40 MSc students.
SFF and cooperating partners (2)

SINTEF
DnV, Telenor, ...
…> 20 Norw. Compan., partly in PROFIT

SE (UiO/NTNU)

SFF

Int’l contacts
ISERN network
Other projects

NTNU T.heim
Six themes:

<table>
<thead>
<tr>
<th>Theme</th>
<th>Responsible(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object-oriented development and maintenance</td>
<td>D. Sjøberg</td>
</tr>
<tr>
<td>Incremental and component-based development</td>
<td>R. Conradi, D. Sjøberg</td>
</tr>
<tr>
<td>Methods to achieve quality of Web-based, distributed systems</td>
<td>L. Jaccheri</td>
</tr>
<tr>
<td>Estimation, planning and risk evaluation of software projects</td>
<td>M. Jørgensen</td>
</tr>
<tr>
<td>General software process improvement and quality work</td>
<td>R. Conradi, T. Skramstad</td>
</tr>
<tr>
<td>Empirical software engineering</td>
<td>M. Jørgensen, T. Stålhane</td>
</tr>
</tbody>
</table>

Research method: Model construction and subsequent validation in industry, among students and through international cooperation.