Software Engineering and Scientific Computing

Barbara Paech, Hanna Valtokari
Institute of Computer Science
Im Neuenheimer Feld 326
69120 Heidelberg, Germany
http://se.ifi.uni-heidelberg.de
paech@informatik.uni-heidelberg.de
<table>
<thead>
<tr>
<th>Time</th>
<th>Topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>9:00</td>
<td>Modeling</td>
</tr>
<tr>
<td></td>
<td>Knowledge Management</td>
</tr>
<tr>
<td>10:30</td>
<td>Break</td>
</tr>
<tr>
<td>11:00</td>
<td>Process models</td>
</tr>
<tr>
<td></td>
<td>Scientific Software Engineering</td>
</tr>
<tr>
<td>12:00</td>
<td>Lunch</td>
</tr>
<tr>
<td>13:00</td>
<td>Tools, Exercises</td>
</tr>
<tr>
<td></td>
<td>Incl. a short break</td>
</tr>
<tr>
<td></td>
<td>Branches and Tagging in Subversion</td>
</tr>
<tr>
<td></td>
<td>IDE</td>
</tr>
<tr>
<td></td>
<td>Wrap-Up, Feedback</td>
</tr>
<tr>
<td>16:00</td>
<td>End</td>
</tr>
</tbody>
</table>
In this course: Programming in a small team

- **Development**
  - Context
  - Requirements Engineering
  - Architecture
  - Design
  - Implementation
  - Version management

- **Quality management**
  - Product (Testing, Inspection, Metrics)
  - Process (Metrics, Improvement)

- **Evolution**
  - Enhancement
  - Re-use
  - Re-engineering
  - Change management

- **Project management**
  - Team
  - cost
  - schedule
  - Risks
  - Customer/Contractor

**Documentation Knowledge management**
Programming in a small team

What is Ron doing?

What is Ron doing?

Project management
Issue Tracking

I want to change Ginnys code

Version management, Build management

I want to check Harrys changes

I want to check Harrys changes

Quality assurance
Testing

I want to explain my ideas to Hermione

I want to explain my ideas to Hermione

Modeling
Knowledge Management

Modeling
Knowledge Management
Documentation, Modeling
Documentation

- Problem description
- Contract
- Customer requirements
- Software specification
- Architecture definition
- Subsystem specification
- Component specification
- Code
- Acceptance Test plan
- Usage test plan
- System Test plan
- Integration Test plan
- Component Test plan
Documents needed for Communication

- Important if
  - Long-time use of the SW
  - System complex
  - Many project participants
  - Decisions have legal relevance

- Software documents typically text and models
Typical Document Structure

1. Introduction
1.1 Purpose
   - Who has created the document how?
   - Who should read the document why??
   - Whoy has to adhere to the content?
1.2 Executive Summary
   - Main content
1.3 Definitions and Abbreviations
   - Including glossary
1.4 References, Standards and Directives
1.5 Overview
   - Content and Structure of the document

2. – X. Main Content

X+1. Conclusion
X+2. Appendices
Example Requirements Document

2. Product Context

2.1 Purpose of the Product
   Business goals

2.2 Stakeholder
   Everyone who is participating in, interested in, affected by the product

2.3 Context Processes
   Business processes which involve the product

3. System Requirements

3.1 Main Features

3.2 Architectural constraints

3.3 Functional requirements
   Use Cases, system functions, GUI

3.4 Non-functional requirements

4. Project requirements

4.1. Assumptions and Dependencies (incl. Risks)

4.2. Product Acceptance
How much and how much detail?

- Depends on the risk, if the readers of the documents do not find the necessary information

- … if it is difficult for the readers to ask the authors personally
  - E.g. distributed development

- … if there is a high risk involved when the software has bugs
  - E.g. life-critical software

- … if there is a high probability of later changes
  - E.g. context changes (new business processes)
Good Documents should be …

- Understandable for all readers
  - Non-ambiguous, complete, consistent

- Understandable for all readers relying on the information, e.g. for requirements documents
  - Customer:
    - Correct: are these really the customer requirements?
  - Requirements engineering: assess progress, ensure consistent change
    - Prioritized, traceable, changeable?
  - Developer:
    - Realizable?
  - Tester:
    - Can test cases be derived?
Characteristics of a model

- A model is essentially a **representation of an original, which is reduced in size or abstracts from details** [Stachowiak 73]

- A model is a system abstraction **with the purpose to support thinking about the system (leaving out details)** [Brügge 00].

- 3 important model characteristics
  - **mapping** (there is an original)
  - **reduction** (original is not represented completely)
  - **pragmatic** (model should be used instead of the original within in a specific context, purpose)

- **Note**: model can be **pre-scriptive or de-scriptive**
Example model

Model 1

Model 2

Model 3
Notation (Language) for a model

- Everything you need to know to create and use models

- Syntax
  - Symbols allowed
  - Abstract Syntax (main concepts)
  - Well-formedness conditions

- Semantic
  - How to interpret the model

- Pragmatics (usage methodology)
  - Techniques for analysis (e.g. type checking, consistency checks)
  - Techniques for simulation
  - Techniques for transformation (e.g. Refactoring)
  - Techniques for generation
Models in software development

- Context, Usage, System
- Requirement
- Architecture
- Implementation
- Interpretation
- Abstraction
- Model
- Views
- Communication
- Analysis
- Is part of System documentation

Barbara Paech, Hanna Valtokari
© 2010 Institut für Informatik, Ruprecht-Karls-Universität Heidelberg
Modeling techniques

SA = Structured Analysis
SD = Structured Design
ER = Entity-Relationship
OMT = Object Modeling Technique
SSADM = Structured Systems Analysis and Design Method
Modeling

- Context
- Requirements Engineering
- Architecture
- Design
- Implementation

Rationale

- Text Activity Diagram
- Structured Text, Use Case
- Entity-Relationship-Diagram
- Deployment diagram, Component diagram
- Class diagram, Object diagram, Interaction diagram, state diagram
- Programming language

Unified Modeling Language
Unified Modeling Language (UML)

- Since 1997 De-Facto-Standard in industry (www.uml.org)
- Mainly for object-oriented development
- defines
  - Structure diagrams (System statics)
  - Behaviour diagrams (System dynamics)
Structure diagrams UML

- **Design**
  - Class diagram
  - Object diagram (complex situations)
  - Package diagram (sets of classes)

- **Architecture**
  - Composition diagram (internal and external interface)
  - (logical) component diagrams
  - Deployment diagram (physical components distributed)
Behaviour diagrams UML

- **Processes**
  - Use Case Diagrams (overview of system functionality)
  - Activity diagrams (sequences of activities)
  - State diagrams (sequences of states)

- **Interaction**
  - Sequence diagram (sequences of messages)
  - Communication diagrams (focus on one component)
  - Timing diagrams (Communication between automata)
  - Interaction overview diagram
Goal structure

Goal

System

Actor

Interaction

Message

Service

Data

Activity

Process

State

Role

is Partner of

has

has

Encapsulates

Encapsulates

performs

sends, receives

knows

Barbara Paech, Hanna Valtokari

Vorlesung – SE and SC – SS 2010/11
© 2010 Institut für Informatik, Ruprecht-Karls-Universität Heidelberg
## Modeling technique focus

| Data / State | Structure Changes | Entity/Relationship diagram  
|             |                   | State diagram |
| Activity / Process | Structure Sequences | Data flow diagram  
|                   |                   | Aktivity diagram, Petri net |
| Service | Structure Behaviour | (Use case diagramm)  
|         |                   | State diagram |
| Goal | Structure | Goal structure diagram |
| Role | Definition Control state changes | Class diagram  
|       |       | State diagram |
| Message / Interaction | Communication structure Sequences | Use Case Text  
|                     |                   | Object model  
|                     |                   | Sequence, Communication diagr. |
- What documents do you create or use?
- What documents could you create or use?
- What modeling techniques do you use
- What modeling techniques could you use?
- Ch. Rupp, S. Queins, B. Zengler, „UML 2 glasklar“, Hanser Verlag 2007
Knowledge Management
Motivation Knowledge Management

Why this one?

- Documents mostly contain the final decision
- Discarded options and criteria are not documented
- => Decisions
  - Do not reflect all important criteria
  - Are not convincing for others
  - Get overthrown (and dead ends are entered again)
# Software Engineering Knowledge

<table>
<thead>
<tr>
<th>Product Knowledge</th>
<th>Knowledge of the System</th>
<th>Knowledge on the process (Roles, Activities, Dokuments)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Content:</strong></td>
<td>Specification, Design, Code, Test plan, etc</td>
<td>Content: Project plan, Cost plan, Tasks, Guidelines</td>
</tr>
<tr>
<td><strong>Rationale:</strong></td>
<td>Product Goals, Criteria, Options, Assessments</td>
<td><strong>Rationale:</strong> Project goals, Risk assessment, Criteria, Options</td>
</tr>
<tr>
<td>Organi-Sational Knowledge</td>
<td><strong>Content:</strong> Domain model, System architecture, Design pattern</td>
<td><strong>Content:</strong> Process model, Best Practices, Experiences</td>
</tr>
<tr>
<td><strong>Rationale:</strong></td>
<td>on the level of generalized models (e.g. Forces for patterns)</td>
<td><strong>Rationale:</strong> on the level of generalized models (e.g. sucess factors for best practices)</td>
</tr>
</tbody>
</table>
- Which knowledge do you capture?
- Which knowledge could you capture?
Process models
What to do when?

- Waterfall model
- Evolutionary development
- V-model
- Rational Unified Process
- Agile methods
  - XP
  - Scrum
  - Test driven development
- Simple, but
  - Tangible product too late (too much paper)
  - Quality assurance too late
Evolution / Iteration / Increment

- Core Idea: early validation of requirements

<table>
<thead>
<tr>
<th>Method</th>
<th>Validation Of requ.</th>
<th>Operational use</th>
<th>Defined stages</th>
<th>Operational use of parts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rapid Prototyping</td>
<td>Through prototype</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Evolutionary</td>
<td>Continue with prototype</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Iterative</td>
<td>Define Iterations (comprise implementation of parts of requ.)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Incremental</td>
<td>Step-wise development of a core system</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- But: Often difficult to manage especially if
  - No stable architecture
  - No general plan
Core idea: early quality assurance

but

• Does not reflect evolution

Rational Unified Process

But:
often too much paper work,
to little flexibility

[http://www-01.ibm.com/software/awdtools/rup/]
Extreme Programming Principles

1. Fast Feedback
   • The learning process depends on the time between the activities.

2. Straightforward Thinking
   • Simple solutions are often sufficient

3. Incremental changes
   • Do not change everything at once, but instead in small steps

4. Embrace change
   • Do not fear changes – it induces more costs to postpone changes

5. Quality focus
   • Quality supports the flexibility to react to changes
XP Practices

Project life-cycle:
- Acceptance Test
- Common Ownership

Development life-cycle:
- Unit Test
- Simple Design
- Refactoring
- On-Site Customer
- Pair Programming
- Metaphor
- Short Increments
- Continuous Integration
- Planning Game
- 40 hour week
- Programming Standards

Supporting Practices:
- Continuous Integration
- Metaphor
- Simple Design
- Pair Programming
- Unit Test
- Refactoring
- Acceptance Test
- Planning Game
- 40 hour week
- Programming Standards
- Common Ownership

Customer Development life-cycle:
- Project life-cycle:
Scrum

User stories

Analysis: WHAT

Estimation, Prioritization

Design: HOW

Metrics, experiences

Implementation, Evaluation

Vision!

Sprint Planning 1

Selected Product Backlog

Sprint Planning 2

Product Backlog

Sprint Backlog

Retrospective

New Functionality

Sprint
Scrum ideas

- Change Management, **Focus on Team**

**Main idea:**
  - Develop software in sprints
  - Daily meetings: Daily Scrum
  - Team is responsible for planning and results

**Roles:**
  - **Product Owner (Customer, User, Management)**
    - Vision, Prioritization
  - **Team**
  - **ScrumMaster (not Project manager!)**
    - Supports Team
    - Moderates between Product-Owner and Team
Scrum Key Practices

- **Sprint planning meeting** held at the beginning of each iteration
  - Analyze and prioritize current product backlog
  - Select overall goal for sprint, Decide how to achieve the goal (design)
  - Create a sprint backlog from the product backlog (more about this below)
  - Estimate backlog in hours
    - Nothing should be longer than a couple of working days
    - Anything that is should be broken into smaller testable/deliverable chunks

- **Daily scrum meeting**
  - Every morning, 15 minutes long, standing up (to make sure it stays 15 minutes long)
  - Everyone says:
    - What they did yesterday, What they are going to do today, What stands in their way
  - *Not* a status update, but rather making commitments to colleagues

- **Sprint review** held at the end of the iteration
  - Team presents what it accomplished
    - Demo, not slides, And yes, everything can be demo'd

- **Sprint retrospective** also held at the end of the iteration
  - What do we want to start doing?
  - What do we want to stop doing?
  - What do we want to keep doing?
Test driven development

- Main idea:
  - write test cases first
  - continuous testing
Which process model do you use?
Which process model could you use?
Scientific Software engineering
Kinds of scientific software

- **Scientific workflow**
  - Software to automate a process of performing a big experiment or data analysis
  - Describe the structure of the process (workflow)
  - Support the semi-automatic execution (workflow management)

- **High performance computing**
  - Complex simulation on parallel computers

- **Framework, library**
  - Code from which algorithms for a specific problem can be created/adapted
Problem sources

Phänomen  Algorithmus  Modell  Software

\[ x = y \times z \]

Mögliche Fehlerquellen [1]
Quality assurance for scientific software

- 1. Check Code
  - Coverage
  - Use known test oracles

- 2. Verify Algorithm
  - Check accuracy of the result
  - Often grid convergence testing
  - Invariants

- 3. Scientific Validation
  - Compare results with experiment results
Conclusion

- Dare to do some steps in Software Engineering
  - You can only judge their value, if you tried some out
- Talk to other people about it
  - You can learn a lot from your colleagues (in other groups)