Software Life Cycles and Configuration Management

Lecture 11

Software Engineering
TDDC88/TDDC93
autumn 2008

David Broman
Department of Computer and Information Science
Linköping University, Sweden
davbr@ida.liu.se

Theory Lecture Plan

L1 - Course Introduction and Overview
L2 - Project Management
L3 - Requirements
L4 - Acceptance Testing and Quality Factors
L5 - UML
L6 – Design Patterns
L7 – System design and architecture
L8 - Testing Theory
L9 - Testing in Practice
L10 - Inspection
L11 - Software Life Cycles and Configuration Management
L12 - Software Quality Management
L13 - Course Summary, Exam examples, Questions

Part I
Life Cycles and Process Models

Part II
Methodologies and Processes - RUP, XP, and Scrum

Part III
Software Configuration Management
A Software Life-cycle Model
Which part will we talk about today?

Part I
Life Cycles and Process Models

Part II
Methodologies and Processes - RUP, XP, and Scrum

Part III
Software Configuration Management

Agenda - What will you learn today?

Part I
Life Cycles and Process Models

Part II
Methodologies and Processes - RUP, XP, and Scrum

Part III
Software Configuration Management
Part I
Life Cycles and Process Models

---

**Project vs. Process**

**Project**
- Start and stop
- Goal
- An orderer
- A budget
- A single-time occurrence

**Process**
- Ordered set of activities
- May contain subprocesses
- Goal of each activity
- Each activity has entry/exit criteria and input/output.
- Processes are reoccurring
- Constraints
Software life-cycle model (Repetiation L1)

Software Life-Cycle
(Software Engineering Process)

Idea

Activity 1

Activity 2

Activity 3

Software Product

Stockholm Subway Map

A software life-cycle model
(Software Engineering Process Model)

Very Complex Software Process

Part I
Life Cycles and Process Models

Part II
Methodologies and Processes - RUP, XP, and Scrum

Part III
Software Configuration Management

Model of a life-cycle (a Process Model)

Carol the customer

Diana the developer

Time
A familiar model?

- **Requirements**
  - Validate Requirements, Verify Specification
- **System Design** (Architecture, High-level Design)
  - Verify System Design
- **Module Design** (Program Design, Detailed Design)
  - Verify Module Design
  - Verify Implementation
- **Implementation** of Units (classes, procedures, functions)
  - Unit testing

**Part I**
Life Cycles and Process Models

**Part II**
Methodologies and Processes - RUP, XP, and Scrum

**Part III**
Software Configuration Management

The V-model

- **Requirements**
  - Validate Requirements, Verify Specification
- **System Design**
  - Verify Design
  - Feedback and iterations are possible
- **Module Design** (Program Design, Detailed Design)
  - Integration Testing
  - Module Testing (Integration testing of units)
- **Implementation** of Units (classes, procedures, functions)
  - Unit testing

**Part I**
Life Cycles and Process Models

**Part II**
Methodologies and Processes - RUP, XP, and Scrum

**Part III**
Software Configuration Management
Another model...

The Waterfall model

- One of the first life-cycle models (Royce, 1970)
- Very common, very criticized

Finish each phase before continue to next.

Why is the waterfall model so criticized?
Which are the problems?
Can it be useful sometimes?

Milestone and deliverable at each step. (Artifacts such as Design document, Req. Specification, etc.).
The Waterfall model - some arguments

**Pros**
- Simple, manageable and easy to understand
- Fits to common project management practices (milestones, deliverables etc.)
- Focus on requirements and design at beginning, save money and time at the end
- Can be suitable for short projects (some weeks)
- Can be suitable for "stable" projects, where requirements do not change
- Focus on documents, saves knowledge which can be reused by other people.
- Widely used, e.g. US Department of Defense
- Can be suitable for fixed-price contracts

**Cons**
- Software requirements change, hard to sign-off on a SRS.
- Early commitment. Changes at the end, large impact.
- Feedback is needed to understand a phase. E.g. implementation is needed to understand some design.
- Difficult to estimate time and cost for the phases.
- Handling risks are not part of the model. Pushes the risks forward.
- Software "is not" developed in such a way. It evolves when problems are more understood. Little room for problem solving.
Can we improve the model?

- Requirements
- System Design
- Program Design
- Implementation
- Integration Testing
- System Testing
- Acceptance Test
- Maintenance

Iteration back to previous phase

Danger! E.g. a performance problem can result in a major requirements change. Very expensive rollback...

Part I
Life Cycles and Process Models

Part II
Methodologies and Processes - RUP, XP, and Scrum

Part III
Software Configuration Management

Do it twice?

First round, a prototype

Second round, do it right.

Input to the phases in the second round

The original paper is actually misunderstood!
- Iteration of phases
- "Do it twice" prototype

(Royce, 1970) includes /square4
Is overlapping phases a solution?

When do we "sign-off", e.g. when do we have all requirements?

What if a major design flaw is discovered at the testing phase?

When do we "sign-off", e.g. when do we have all requirements?

What kind of structure have we actually achieved? No sign-off. How does this help us?
What should be built?

"The hardest single part of building a software system is deciding precisely what to build”

(Frederick P. Brooks)

Iterative Development

When should the releases take place?

Time-boxing - The time period is fixed for each iteration.

What should be included in the release?

Prioritized functionality - Do the most important parts first.

Customer Feedback

Final Release!
Dependent project parameters (revisited)

Calendar time and resources are fixed

Select the most important functions

Select quality. E.g. how general should we be?

- Calendar Time
- Resources
- Features
- Quality

Prioritization - some matrices

Customer Benefit

High
Low

Importance

High
Low

Sweet Spot
Avoid

Development Effort

Low
High

Urgency

Low
High

Sweet Spot
Avoid
Iterative vs. Incremental Development

**Incremental Development**
Add a new "part" at each increment

**Iterative Development**
Improve a "working system" at each iteration

---

Spiral Model

**Determine goals, alternatives, and constraints**

**Evaluate Alternatives and risks**

Main feature: RISKS

*RA = Risk Analysis*

*P1 = Prototype 1*

*CO = Concepts of operation*

*BAC = Budget, Alternatives, Constraints*

Plan Next Phase
Iterative Development - Cons

Is iterative development the silver bullet?

- Problem with current business contracts, especially fixed-price contracts.
- With small iterations, can be hard to map customer requirements to iterations.

Customer Feedback

Iterative Development - Pros

Pros

- Misunderstandings and inconsistency are made clear early (e.g. between requirement, design, and implementation)
- Encourage to use feedback -> elicit the real requirements
- Forced to focus on the most critical issues
- Continuous testing offers a project assessment
- Workload is spread out over time (especially test)
- The team can get "lesson learned" and continuously improve the process
- Stakeholders gets concrete evidence of progress
Part II
Methodologies and Processes - RUP, XP, and Scrum

We are using an iterative process!

Define a plan with 1..N iterations. We do not have to care about plans...

Now, let’s hack!

Is this a good iterative process? Of course not. We need some structure!

Methodologies and defined Processes
Part I
Life Cycles and Process Models

Part II
Methodologies and Processes - RUP, XP, and Scrum

Part III
Software Configuration Management

Processes, Models, Methodologies...

Process Models

Waterfall model
V-model
Prototype model
Spiral model

"what" at a high level of abstraction

Which is the "best" approach?

Rational Unified Process (RUP)

Extreme Programming (XP)
Scrum

agile methods

"what" and to a certain level "how"

Question:
What is the difference between a methodologist and a terrorist?

Answer:
You can negotiate with a terrorist.

Processes, Models, Methodologies...

Process Models

"what" at a high level of abstraction

Methodologies and defined Processes

"what" and to a certain level "how"
Goals with a software development process

- Guidance about order and content of team activities.
- Specify when and which artifact that should be produced.
- Direct individual developers' tasks and the team as a whole.
- Give criteria for monitoring and measuring activities and generated products.

What is Rational Unified Process (RUP)?

A software engineering process
A disciplined approach to produce high quality software

A process product
- A software (web based) offered by IBM (earlier Rational)
- OpenUP/Basic, an lightweight open source variant

A process framework
- Adapted and extended to suit a specific organization
Part I
Life Cycles and Process Models

Part II
Methodologies and Processes - RUP, XP, and Scrum

Part III
Software Configuration Management

---

RUP - Elements of the Process

Roles: the who
- e.g. system analyst, designer, test designer

Activities: the how

Artifacts: the what
- e.g. use-case models, source code, documents

Workflows: the when
- Core workflow to each discipline

- Receive Order
- Get item from stock
- Express Delivery
- [prioritized] [else]
- Delivery via mail
- Close Order
- Send Invoice
- Receive Payment

Disciplines

The "container" for the four elements: roles, activities, artifacts and workflows

- Core Technical Disciplines
- Business Modeling
- Requirements
- Analysis and Design
- Implementation
- Test
- Deployment
- Change & Config. Mgm.
- Project Mgm.
- Environment.

- Core Supporting Disciplines

---

Part I
Life Cycles and Process Models

Part II
Methodologies and Processes - RUP, XP, and Scrum

Part III
Software Configuration Management
### Part I: Life Cycles and Process Models

#### RUP - Phases and Milestones

**Inception**
- Formulate scope
- Capture most important requirements
- Plan, risk, staffing, project plan
- Synthesize a candidate architecture
- The project may be cancelled after this phase similar to a "Pre-study"

**Elaboration**
- Define architecture
- Specify requirements more precisely
- Executable architecture prototype
- Define project plan

---

#### Elaboration

*Life-cycle architecture milestone*

---

#### Inception

*Life-cycle objective milestone*
Part I
Life Cycles and Process Models

Part II
Methodologies and Processes - RUP, XP, and Scrum

Part III
Software Configuration Management

**RUP - Phases and Milestones**

**Construction**
- Resource management and control
- Design, Implementation, and Testing
- Output (software + documentation) ready for users.

Initial Operational Capability milestone (beta-release)

**Transition**
- Transition of the product to users
- Beta-testing
- Training of users and maintainers
- Rollout of the product to operational environment

Product release milestone
Part I
Life Cycles and Process Models

Part II
Methodologies and Processes - RUP, XP, and Scrum

Part III
Software Configuration Management

---

RUP- Phases and Milestones

Was not RUP iterative???

Internal milestones and releases

Iterations within phases

Time

Inception (10%)
Elaboration (30%)
Construction (50%)
Transition (10%)

---

Disciplines and Phases

Business Modeling
Requirements
Analysis and Design
Implementation
Test
Deployment
Change & Config. Mgm.
Project Mgm.
Environment.

---

Part I
Life Cycles and Process Models

Part II
Methodologies and Processes - RUP, XP, and Scrum

Part III
Software Configuration Management
Agile Approaches - Agile Alliance

Lightweight approaches to satisfy the customers with “early and continuous delivery of valuable software”

Manifesto for Agile Software development

Favor

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

(http://agilemanifesto.org, 2001)

Extreme Programming - Values and Principles

A lightweight methodology for vague or rapidly changing requirements

Communication

Mutual benefit "win-win", automated testing

Reflection

"How" and "why" are we working

Redundancy

If it fails. E.g. pair programming.

Simplicity

Feedback

Changes need feedback

Courage

"If you know what the problem is, do something"

Respect

Baby steps

“What is the least that you can do that can be shown to be in the right direction?”
### Extreme Programming - Some Practices

**Pair Programming**
- Focus on task
- Clarify ideas
- Rotate frequently

**Continuous Integration**
- Integrate and test often
- Automated build system
- Automated regression tests (e.g. JUnit)

**Refactoring**
- Behavior preserving transformation
- Tool support, e.g. Eclipse

**Stories**
- "requirements", but not mandatory
- Name + short story
- On index cards (paper)

**Test-First Programming**
- Create tests before code
- Focus on interface and "what is needed"
- Gets tests for free

---

### Scrum

**Approach public in 1996 at OOPSLA**
(Ken Schwaber, Jeff Sutherland)

"Scrum" strategy used in rugby for getting and out-of-play ball back into play.

---

---
**Scrum Overview**

- **Product backlog**
  - All requirements
  - Prioritized only by product owner
  - Never finalized
  - From anyone (users, customers, sales)

- **Sprint backlog**
  - What to do in one sprint

- **Sprint planning meeting**
  - Plan what to do in the next sprint
    (sprint = one iteration)

- **Sprint**
  - 30 - days iteration
  - product increment

- **Daily Scrum**
  - Short (15 min)
  - done since last meeting, todo, problems
  - Pick from sprint backlog

- **Scrum Master**
  - Enforce scrum practices

- **Executable Product Increment**

---

**Part III**

**Software Configuration Management**
What is configuration management?

Configuration Item Identification
- Source code modules
- Test scripts
- Design documents
- Build systems

SCM

Baseline - a "snap-shot" of configuration items. Normally reviews in some way.

How do we control changes?

Part I
- Life Cycles and Process Models

Part II
- Methodologies and Processes - RUP, XP, and Scrum

Part III
- Software Configuration Management
What is configuration management?

Configuration Item Identification
- Source code modules
- Test scripts
- Design documents
- Build systems

Configuration control
- Only authorized people may make changes
- Larger changes - change requests

SCM

Change Control Board (CCB)
- Make change decisions.
- Only large changes (e.g. new major requirements)

CHANGE REQUEST
Project:__________
Classification:_____
Priority:__________
Date:___________
Change Description:

Auditing
- Ensure that the items are complete and consistent.
- Make sure that the configuration is tested and meets requirements.

Status accounting
- Document and report changes to those involved.
 SCM tools

Workflow systems
- Define processes for change requests

Change report system
- Bug-tracking
- New features
- Change request
  (e.g. free alternatives: Bugzilla, Trac)

Locked checkout / no locks

Commit - add comments
History - Find where the fault was added
Blame - Who added a certain code line
Diff - What has changed between two versions?

Tool examples:
- Clear Case,
- Visual Source Safe
- Perforce,
- CVS
- Subversion

Version handling example

Development Branch(s)
- v1.1.24.1 → v1.1.24.2
- Test
- Merge

Trunk
- v1.0.23 → v1.0.24 → v1.0.25 → v1.0.26

Release Branch(s)
- v1.2.24

Part I
Life Cycles and Process Models

Part II
Methodologies and Processes - RUP, XP, and Scrum

Part III
Software Configuration Management
Sync- and Stabilize (e.g. Microsoft)

Implement new stuff

Commit to SCM repository

Daily build
(automatic compile and link)

Fail? Success?
(Report via e-mail, red-lamp etc.)

Smoke test
(automatic test of system)

Stub-out e.g. user interfaces and databases

Automatic blame :-)
Further reading (Books)

### Rational Unified Process (RUP)

### Agile, Extreme Programming, Scrum

### Configuration Management

Further reading (Web)

### Extreme Programming
- Extreme Programming, a gentle introduction