Software Life Cycles, Processes, and Agile Methodologies

Lecture 4

Software Engineering
TDDC88/TDDC93
autumn 2013

Aseel Berglund (slides by David Broman)
Department of Computer and Information Science
Linköping University, Sweden
aseel.berglund@liu.se
Knowledge Areas
Week 36 – Requirements
Week 37 – Planning and Processes
Week 38 – Design and Architecture
Week 39 – Testing and SCM
Week 40 – Software Quality

Part I
Life Cycles and Process Models

Part II
Agile Methodologies

Part III
RUP
A Software Life-cycle Model

Part I
Life Cycles and Process Models

Part II
Agile Methodologies

Part III
RUP

- Requirements
- System Design (Architecture, High-level Design)
- Module Design (Program Design, Detailed Design)
- Implementation of Units (classes, procedures, functions)
- Unit testing
- Module Testing (Integration testing of units)
- System Testing (Integration testing of modules)
- Acceptance Test (Release testing)
- Maintenance

Steps:
- Validate Requirements, Verify Specification
- Verify System Design
- Verify Module Design
- Verify Implementation

Support:
- Project Management, Software Quality Assurance (SQA), Supporting Tools, Education

Assessment:
- Validate Requirements
- Verify Specification
- Verify System Design
- Verify Module Design
- Verify Implementation

Agenda - What will you learn today?

Part I
Life Cycles and Process Models

Part II
Agile Methodologies

Part III
Rational Unified Process, RUP
Part I
Life Cycles and Process Models
Project vs. Process

**Project**

A project is a temporary endeavor undertaken to create a unique product or service

---

**Part I**
Life Cycles and Process Models

**Part II**
Agile Methodologies

**Part III**
RUP
Process

- Ordered set of activities
- May contain sub-processes
- Goal of each activity
- Each activity has entry/exit criteria and input/output
- Constraints

Activities:
1. Activity 1
2. Activity 2
3. Activity 3

Processes are reoccurring
Software life-cycle model

representation or simplified version of a concept, phenomenon, relationship, structure, system, or an aspect of the real world.
Model of a life-cycle (a Process Model)

Carol the customer

Diana the developer

Time

Abstraction

Part I
Life Cycles and Process Models

Part II
Agile Methodologies

Part III
RUP
A familiar model?

- **Requirements**
- **System Design** (Architecture, High-level Design)
- **Module Design** (Program Design, Detailed Design)
- **Implementation** of Units (classes, procedures, functions)
- **Unit testing**
- **Module Testing** (Integration testing of units)
- **System Testing** (Integration testing of modules)
- **Acceptance Test** (Release testing)
- **Maintenance**

**Validation Phases:**
- Validate Requirements, Verify Specification
- Verify System Design
- Verify Module Design
- Verify Implementation

**Life Cycles and Process Models:**
- **Part I**
  - Life Cycles and Process Models
- **Part II**
  - Agile Methodologies
- **Part III**
  - RUP
The Waterfall Model

- One of the first life-cycle models (Royce, 1970)
- The waterfall development model originates in the manufacturing and construction industries
- Very common, very criticized
The Waterfall model

Requirements → System Design → Module Design → Implementation → Unit Testing → Module Testing → System Testing → Acceptance Test → Maintenance

Finish each phase before continue to next.

Milestone and deliverable at each step. (Artifacts such as Design document, Req. Specification. etc.).

Time

Part I
Life Cycles and Process Models

Part II
Agile Methodologies

Part III
RUP
The Waterfall Model

- Why is the waterfall model so criticized?
- Which are the problems?
- Can it be useful sometimes?
The Waterfall Model - some arguments

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.
Pros

- Simple, manageable and easy to understand
- Fits to common project management practices (milestones, deliverables etc.)
- 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.
- Can be suitable for fixed-price contracts
Can we improve the model?

Danger! E.g. a performance problem can result in a major requirements change. Very expensive rollback…
Do it twice?

First round, a prototype

Second round, do it right.

The original paper is actually misunderstood!
(Royce, 1970) includes
- Iteration of phases
- "Do it twice" prototype

Input to the phases in the second round
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 if a major design flaw is discovered at the testing phase?

Release!

Time

- requirements
- design
- implementation
- test

Part I
Life Cycles and Process Models

Part II
Agile Methodologies

Part III
RUP
What should be built?

"The hardest single part of building a software system is deciding precisely what to build”  
(Frederick P. Brooks)

What should be built?

The hardest single part of building a software system is deciding precisely what to build.  
(Frederick P. Brooks)

How? By delivering several releases?
**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**

**R1**

**R2**

**Final Release!**

---

**Iteration 1**

**Iteration 2**

**Iteration 3**

---

**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**

**R1**

**R2**

**Final Release!**

---

**Iteration 1**

**Iteration 2**

**Iteration 3**

---

**Customer Feedback**

**R1**

**R2**

**Final Release!**

---

**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**

**R1**

**R2**

**Final Release!**

---

**Iteration 1**

**Iteration 2**

**Iteration 3**

---

**Customer Feedback**

**R1**

**R2**

**Final Release!**

---

**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**

**R1**

**R2**

**Final Release!**

---

**Iteration 1**

**Iteration 2**

**Iteration 3**

---

**Customer Feedback**

**R1**

**R2**

**Final Release!**

---

**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**

**R1**

**R2**

**Final Release!**

---

**Iteration 1**

**Iteration 2**

**Iteration 3**

---

**Customer Feedback**

**R1**

**R2**

**Final Release!**

---

**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**

**R1**

**R2**

**Final Release!**

---

**Iteration 1**

**Iteration 2**

**Iteration 3**

---

**Customer Feedback**

**R1**

**R2**

**Final Release!**

---

**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**

**R1**

**R2**

**Final Release!**

---

**Iteration 1**

**Iteration 2**

**Iteration 3**

---

**Customer Feedback**

**R1**

**R2**

**Final Release!**

---

**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.
Dependent project parameters (revisited)

Calendar time and resources are fixed

- Calendar Time
- Resources

Select the most important functions
- Features

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

---

Part I
Life Cycles and Process Models

Part II
Agile Methodologies

Part III
RUP
Prioritization of requirements

- **Customer Value**
  - High
  - Low

- **Development Effort**
  - Low
  - High

- **Sweet Spot**
- **Avoid**
Effort Estimation... a good approach?

How long time does it take for you to implement the encryption layer?

No idea. I have never done this before... I wonder if it is even possible.

8 months +- 2 months
**Expert Judgment - the Delphi technique**

- Experts make individual predictions secretly
- Estimates are shown to each other (anonymously)
- Experts discuss the estimates
- [Convergence] or [No Convergence]

**Algorithmic Methods - COCOMO and COCOMO II**

**COCOMO (Boehm, 1981)**
- An formula where parameters are estimated using real projects.
- Input: No of code lines
- Output: Effort (time)

**COCOMO II**
- Takes into account changes in SE, such as component reuse, prototyping
- Other inputs than number of code lines. E.g. functionality from requirements, number of screens etc.
Is iterative development the silver bullet?

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

Customer Feedback

---

Iterative Development - Cons

---

**Part I**
Life Cycles and Process Models

---

**Part II**
Agile Methodologies

---

**Part III**
RUP

---

Final Release!
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 project assessment
- Workload is spread out over time (especially test)
- The team can get "lesson learned" and continuously improve the process
- Stakeholders get concrete evidence of progress
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!**
Processes, Models, Methodologies...

Part I
Life Cycles and Process Models

Part II
Agile Methodologies

Part III
RUP

Process Models

Waterfall model
V- model
Spiral model
Prototype model

"what" at a high level of abstraction

Which is the "best" approach?

Methodologies and defined Processes

Extreme Programming (XP)
Rational Unified Process (RUP)
Scrum

agile

"what" and to a certain level "how"
Part II
Agile Methodologies
Lightweight approaches to satisfy the customers with "early and continuous delivery of valuable software"

**Manifesto for Agile Software Development**

- **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)
According to the 2011 CHAOS Manifesto from the Standish Group, Agile projects are 3 times more often successful than non-agile projects.
What’s Agile?

Iterative and incremental process approach

- Kanban
- Scrum
- eXtreme Programming (XP)
- Feature Driven Development
- DSDM
- CI (Continuous Integration)
- eXtreme Programming (XP)
- ATDD (Acceptance Test Driven Development)
- TDD (Test Driven Development)
- Other
Scrum
Scrum

Approach public in 1995 at OOPSLA

"Scrum" strategy used in rugby for getting an out-of-play ball back into play.
Small, cross-functional teams

Product split into small, roughly estimated, stories

Iterations - sprints

Continuous improvement and deployment
The Sprint

Sprint end date and deliverable do not change.
The Team

Sprint end date and deliverable do not change
The Scrum Master

Scrum Master

The Team

Sprint end date and deliverable do not change
The Product Owner

Inputs from Executives, Stakeholders, Customers, Users, Team

Scrum Master

The Team

Sprint end date and deliverable do not change
Inputs from Executives, Stakeholders, Customers, Users, Team

Product Owner

Scrum Master

The Team

The Product Backlog

A prioritized list of what is required, features, stories

1
2
3
4
5
6
7

Product Backlog

Sprint end date and deliverable do not change
The Sprint Planning Meeting

Inputs from Executives, Stakeholders, Customers, Users, Team

Scrum Master

The Team

Product Owner

1. A prioritized list of what is required, features, stories
2. Team selects starting at top as much as it can commit to deliver by end of sprint

Product Backlog

Sprint Planning Meeting

Sprint end date and deliverable do not change

Part I
Life Cycles and Process Models

Part II
Agile Methodologies

Part III
RUP
The Sprint Backlog

1. A prioritized list of what is required, features, stories
2. Team selects starting at top as much as it can commit to deliver by end of sprint
3. Sprint Planning Meeting

Inputs from Executives, Stakeholders, Customers, Users, Team

Product Owner

Scrum Master

The Team

Product Backlog

Sprint Planning Meeting

Task Breakout

Sprint Backlog

Sprint end date and deliverable do not change

Every 24 Hours

1-4 Week Sprint

Part I
Life Cycles and Process Models

Part II
Agile Methodologies

Part III
RUP
Sample Taskboard

Part I
Life Cycles and Process Models

Part II
Agile Methodologies

Part III
RUP
The Daily Scrum Meeting

Inputs from Executives, Stakeholders, Customers, Users, Team

Scrum Master

The Team

Product Owner

1. A prioritized list of what is required, features, stories
2. Team selects starting at top as much as it can commit to deliver by end of sprint
3. Task Breakout
4. Sprint Planning Meeting
5. Sprint Backlog
6. 1-4 Week Sprint
7. Every 24 Hours

Sprint end date and deliverable do not change

Daily Scrum Meeting
The Burn Down Charts

Inputs from Executives, Stakeholders, Customers, Users, Team

Product Owner

The Team

Scrum Master

Burn down charts

Daily Scrum Meeting

Every 24 Hours

1-4 Week Sprint

Sprint end date and deliverable do not change

Team selects starting at top as much as it can commit to deliver by end of sprint

Sprint Planning Meeting

Task Breakout

Sprint Backlog

Product Backlog

A prioritized list of what is required, features, stories

1

2

3

4

5

6

7

Part I
Life Cycles and Process Models

Part II
Agile Methodologies

Part III
RUP

aseel.berglund@liu.se
The burn down chart

- Only track hours remaining, not hours worked
- X – days (in Sprint)
- Y – hours remaining
- Remove meeting time, vacation etc. from total available hours
- Update only when PBIs are DONE
- When not done – Undone PBIs
Electronic Taskboard

Part I
Life Cycles and Process Models

Part II
Agile Methodologies

Part III
RUP
The Sprint Review Meeting

**Part I**
Life Cycles and Process Models

**Part II**
Agile Methodologies

**Part III**
RUP

---

**Inputs from Executives, Stakeholders, Customers, Users, Team**

- **Product Owner**
- **Scrum Master**
- **The Team**

---

**Product Backlog**

- **Sprint Planning Meeting**
  - A prioritized list of what is required, features, stories

- **Task Breakout**
  - Team selects starting at top as much as it can commit to deliver by end of sprint

- **Sprint Backlog**
- **Every 24 Hours**
- **1-4 Week Sprint**
- **Burn down charts**
- **Sprint Review Meeting**
- **Daily Scrum Meeting**
- **Sprint End date and deliverable do not change**
- **Finished work**
- **Done?**
The Definition of Done!

• When are we done?
• “No more remaining work”
• Includes testing, documentation etc.
• Possible to ship after each sprint
• Everybody – understand what done means

Tools to support done
• Version handling (SCM)
• Automated build
• Automated tests (Continuous integration)
The Sprint Retrospective

Inputs from Executives, Stakeholders, Customers, Users, Team

1. Product Owner
2. Scrum Master
3. The Team

1. A prioritized list of what is required, features, stories

Task Breakout

Sprint Planning Meeting

Team selects starting at top as much as it can commit to deliver by end of sprint

Sprint Backlog

Sprint end date and deliverable do not change

Sprint review meeting

Finished work

Sprint retrospective

Burn down charts

Daily Scrum Meeting

Every 24 Hours

1-4 Week Sprint

Weekly

The Team

Product Owner

Part I
Life Cycles and Process Models

Part II
Agile Methodologies

Part III
RUP
Definition of the S# meetings

**S4 - Scrum of Scrum of Scrum of Scrums** (e.g. once bi-weekly)
**Participants:** CEO, Prod. chief, R&D manager, Operational manager...

**S3 – Scrum of Scrum of Scrums** (once a week)
**Participants:** Scrum masters, product owners, Prod. chief, R&D manager, senior architectures, release manager.

**S2 – Scrum of Scrums** (e.g. twice a week)
**Participants:** Scrum masters, product owners, senior architectures, release manager.

**S1 – Scrum** (daily)
**Participants:** Scrum master, product owner, scrum team.

Figure source: http://www.scrumalliance.org/community/articles/2007/may/advice-on-conducting-the-scrum-of-scrums-meeting
Test Driven Development
TDD
A definition in Steve McConnell's *Code Complete* divides software into two pieces: **internal** and **external quality characteristics**.

- External quality characteristics are those parts of a product that face its users
- Internal quality characteristics are those that do not
TDD is a technique for improving the software’s internal quality

**Well-written code**
- Good design
- A balanced division of responsibilities
- Without duplication (clones)
- Smooth evolution
- Maintainability
Build it right: TDD

Traditional development cycle:
Design → Code → Test

Test-driven development cycle:
Test → Code → Design

Part I
Life Cycles and Process Models

Part II
Agile Methodologies

Part III
RUP
First we write a test

- We are writing a test. Also, we are making design decisions:
  - We are designing the API—the interface for accessing the functionality we’re testing.
  - The test case that we design will be the first “client” of the functionality that we are going to implement.
  - One of the fundamental lessons in designing an interface is that we only evaluate a design effectively and objectively when we try to use it.
The second step of the TDD cycle is to write just enough code to make the test pass.

You’re satisfying an explicit, unambiguous requirement expressed by a test.
And then we refactor

- Take a step back, look at our design, and figure out ways of making it better - refactoring (Relying on the existing tests to keep us from breaking things while we are at it)
- It is all about keeping your software in good health—at all times.
- Refactoring is about applying refactorings on code in a controlled manner
Refactoring

Definition: Refactoring modifies software to improve its readability, maintainability, and extensibility without changing what it actually does.

- External behavior does NOT change
- Internal structure is improved

Mercutio, do you have a minute?

Just a second, Will. I'm refactoring some of my code.

What does that mean?

It means I'm rewriting it the way it should have been written in the first place, but it sounds cooler.
What is TDD?

- Writing the test before the program
- It leads the developer to
  1. think about “how to use” the component (why do we need the component, what’s it for?)
  2. only then about “how to implement”.
- It’s a testing technique as well as a design technique
  - It results into components that are easy to test.
    It results into components that are easy to enhance and adapt.
- The developer can tell at any time:
  - whether everything still works as it should
  - what exactly does no longer work as it once did
The Rhythm of Test-Driven Development (TDD)

Quickly add a test

Run all tests and see them all succeed

Make a little change

Run all tests and see the new one fail

Refactor to remove duplication
Acceptance Test Driven Development
ATDD
- **Acceptance tests** are indicators of the completion of a requirement or feature.
- When all acceptance tests for a requirement or feature are passing, you know you’re done.
Acceptance TDD helps us keep our product’s external quality on track by giving it the correct features and functionality.

- The system should meet what the customer actually need.
Advantages of ATDD

- **Close collaboration**
  - Seeing concrete, working software
  - Building trust and confidence
  - Customer in control
  - Evolving a shared language

- **Tests as a shared language**
  - Tests as specification
  - Specification by example
- At the start of the iteration:
  - the customer explains the expected functionality to the team
  - the customer prioritizes for business value and urgency
  - the team estimates the effort and cost
- Then the team brainstorms the necessary tasks to implement the functionality, details the estimation and team members select their tasks.

Iterative Software Development
How ATDD?

Prioritized functionality

Write acceptance tests

One iteration with short feedback loop

Execute acceptance tests to understand the progress

24 h

Write unite tests based upon acceptance tests

30 d

Increment of functionality

Execute acceptance tests to verify

Part I
Life Cycles and Process Models

Part II
Agile Methodologies

Part III
RUP
TDD or ATDD?

- TDD and acceptance TDD often go hand in hand.
- On the system level, we run our development process with ATDD; and inside the implementation step of each feature; we employ TDD.
看板

Kanban
The two pillars of the Toyota production system are just-in-time production and automation with a human touch, or autonomination. The tool used to operate the system is kanban.

Taiichi Ohno
Father of the Toyota Production System
看板 - Kanban

- Kanban is a Japanese word that means “visual card,” “signboard,” or “billboard.”
- Toyota originally used Kanban cards to limit the amount of inventory tied up in “work in progress” on a manufacturing floor.
- Kanban is a lean approach to agile software development.
How does Kanban Work?

- **Visualize the workflow**
  - Split the work into pieces, write each item on a card and put on the wall.
  - Use named columns to illustrate where each item is in the workflow.

- **Limit WIP** (work in progress) – assign explicit limits to how many items may be in progress at each workflow state.

- **Measure the lead time** (average time to complete one item, sometimes called “cycle time”), optimize the process to make lead time as small and predictable as possible.
A simple Kanban Board

Source: http://www.crisp.se/gratis-material-och-guider/kanban
Work In Progress, WIP, limits are designed to:

- reduce multitasking
- maximise throughput
- enhance teamwork

Reducing multitasking is beneficial for two primary reasons
20% time is lost to context switching per ‘task’, so fewer tasks means less time lost

(from Gerald Weinberg, Quality Software Management: Systems Thinking)
Performing tasks sequentially yields results sooner.

multi-tasking A, B and C (on the top), delivers A much later, and even C slightly later, than sequentially (on the bottom).
Typical Measurements

- **Cycle time** – Measured from when you started working on it
- **Lead time** – Measured from when the customer ordered
- **Quality** – Time spent bugfixing per iteration
- **WIP** – Average number of “stories” in progress
- **Throughput** – Number of “stories” completed per iteration
  (when using fixed iterations)
Benefits of Kanban

- Eliminate over-production, the #1 waste
- Produce only what is ordered, when ordered, & quantity ordered
- Increase flexibility to meet customer demand
- Competitive advantage by sequencing shipments to customers (what they want, when they want it, in the order they want it!)
eXtreme Programming

XP

WE'RE GOING TO TRY SOMETHING CALLED EXTREME PROGRAMMING.

FIRST, PICK A PARTNER. THE TWO OF YOU WILL WORK AT ONE COMPUTER FOR FORTY HOURS A WEEK.

THE NEW SYSTEM IS A MINUTE OLD AND I ALREADY HATE EVERYONE.

Copyright © 2003 United Feature Syndicate, Inc.
Extreme Programming

- Formulated in 1999 by Kent Beck
- XP is “a light-weight methodology for small to medium-sized teams developing software in the face of vague or rapidly changing requirements.”
XP - Some Practices

**Pair Programming**
- Programming as a collaborative conversation
- Focus on task
- Clarify ideas
- Rotate frequently

**Stories**
- "requirements", but not mandatory
- a token for a piece of system capability to be implemented
- Name + short story
- On index cards (paper)

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

**Refactoring**
- Improve the design of existing code without changing its functionality
- Tool support, e.g. Eclipse

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

---

**Part I**
Life Cycles and Process Models

**Part II**
Agile Methodologies

**Part III**
RUP
XP Values

- **Communication**
  - On-site customer, user stories, pair programming, daily standup meetings, etc.

- **Simplicity**
  - "Do the simplest thing that could possibly work" (DTSTT TCPW) principle

- **Feedback**
  - Unit tests tell programmers status of the system
  - Programmers produce new releases every 2-3 weeks for customers to review

- **Courage**
  - Communicate and accept feedback, throw code away, refactor the architecture of a system
Part III

Rational Unified Process, RUP
The Rational Unified Process (RUP) is an iterative software development process framework

- Defined in 1997 by Grady Booch, Ivar Jacobson and James Rumbaugh
- Recognized to be particularly applicable to large projects with large teams
RUP Phases

- **Inception**
  - Shared understanding of the system with the customer

- **Elaboration**
  - Architecture to build the system

- **Construction**
  - Developing the system

- **Transition**
  - Customer takes ownership of system
Lecture Exercise

Short essay about planning and processes

- 1-2 pages (short and concise)
- Deadline 1pm, Monday September 16, Missed deadline = 0 credit
- **One** document – URKUND complains if the documents are too short
- Write your name, email and personal number
- Read the instructions
- Argue and discuss...
What did you learn today?

Part I
Life Cycles and Process Models

Part II
Agile Methods

Part III
Rational Unified Process, RUP