# Design of Software Systems (Ontwerp van SoftwareSystemen)

5 Software Development Processes

Roel Wuyts 2015-2016

#### Software Process

Set of activities that leads to the production of a software product

- lots of processes exist
- share some fundamental activities

# Development Phases

Testing	Validate the solution against the requirements
Analysis	Model and specify the requirements ("what")
Maintenance	Repair defects and adapt the solution to new requirements
Implementation	Construct a solution in software
Requirements Collection	Establish customer's needs
Design	Model and specify a solution ("how")

### Software Development Process

#### A software development methodology is

- a set of partially ordered steps
- to build, deploy, maintain, ... software

#### Examples:

- Waterfall
- Spiral
- XP (eXtreme Programming)
- UP (Unified Process)
  - RUP (Rational Unified Process)
  - Agile UP

Heavyweight	Customizable	Agile (Lightweight)
e.g., Waterfall	Framework	e.g., eXtreme
model,	e.g., Rational	Programming (XP),
V-Process	← Unified →	SCRUM
	Process (RUP)	

Document driven Elaborate workflow definitions Many different roles Many checkpoints High management overhead Highly bureaucratic Focus on

- indiv./interactions rather than process/tools
  working SW rather than documentation
- customer collaboration rather than contract
- responding to change rather than the plan

### Lightweight vs. Heavyweight Processes



Document driven Elaborate workflow definitions Many different roles Many checkpoints High management overhead Highly bureaucratic Focus on

- indiv./interactions rather than process/tools
  working SW rather than documentation
- customer collaboration rather than contract
- responding to change rather than the plan

# Waterfall Model

#### Characterized by

- Sequential steps (phases)
- Feedback loops (between two phases in development)
- Documentation-driven

#### Advantages

- Documentation
- Maintenance easier

#### Disadvantages

- Complete and frozen specification document up-front often not feasible in practice
- Customer involvement in the first phase only
- Sequential and complete execution of phases often not desirable
- Process difficult to control
- The product becomes available very late in the process



#### V-Model

Like the Waterfall model, it is a linear model that is very rigid

- Requirements are expected not to change
- Due to the V-shape, the first tests are the implementation tests

Unlike the waterfall model, every integration is tested



## Lightweight vs. Heavyweight Processes



# UP: Iterative and Incremental development

iterative & incremental development : embracing change

– Essential for SW Development

```
("No Silver Bullet", Brooks, 1987)
```

iterative models: can be iterative w.r.t. value and/or requirements

# Iterative Development (a.k.a. incremental models)

More functionality with each release (new increment)

- Operational quality portion of product within weeks

Non-incremental models (e.g. Waterfall)

- Operational quality complete product at end



# Incremental development (a.k.a. Evolutionary Models)

New versions implement new and evolving requirements



# UP is Use-Case-Driven

Use cases are concise, simple, and understandable by a wide range of stakeholders

 End users, developers and acquirers understand functional requirements of the system

Use cases drive numerous activities in the process:

- Creation and validation of the design model
- Definition of test cases and procedures of the test model
- Planning of iterations
- Creation of user documentation
- System deployment

Use cases help synchronize the content of different models

# UP's 4 Project Life Cycle Phases



#### Inception

- Approximate vision
- Business case
- Scope
- Vague estimates
- Continue or stop?

#### Elaboration

- Identification of most requirements
- Iterative implementation of the core architecture
- resolution of high risks

# UP's 4 Project Life Cycle Phases (ctd)



Construction

#### Transition

- Iterative implementation of lower risk elements
- Beta tests
- Deployment

Preparation for deployment

#### **Iterations and Milestones**

Inception	Elaboration		Construction		Transition		
Preliminary Iteration	lter. #1	Iter. #2					
	Μ	ilestone		Release	Ì	<u>Final pro</u> rele	oduction ase

Each phase concludes with a well-defined milestone.

### **Iterations and Milestones**



Phases consist of one or more iterations

- short fixed-length mini-projects (2 to 6 weeks)
- shift tasks to future iterations if necessary ...
- an iteration represents a complete development cycle
- the end of each iteration is a minor release, a stable, integrated executable subset of the final product

#### The UP Disciplines



#### Advantages

- Incremental & Iterative
- Sits in between heavyweight and agile processes
  - best of both worlds ?
- Customizable
- Potential pitfalls
  - Use Cases do not model all requirements
  - Hard to make really lightweight, even when customized
    - Quite some documentation and process remains

### Lightweight vs. Heavyweight Processes

Heavyweight	Customizable	Agile (Lightweight)
e.g., Waterfall	Framework	e.g., eXtreme
model,	e.g., Rational	Programming (XP),
V-Process	← Unified →	SCRUM
	Process (RUP)	

Document driven Elaborate workflow definitions Many different roles Many checkpoints High management overhead Highly bureaucratic Focus on

- indiv./interactions rather than process/tools
  working SW rather than documentation
- customer collaboration rather than contract
- responding to change rather than the plan

#### Agile Development

#### Group of iterative and incremental software methodologies

#### The Agile Manifesto

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

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

That is, while there is value in the items on the right, we value the items on the left more.

# Extreme Programming (XP)

Point of XP: coping with change and uncertainty

Based on number of practices:

- small, frequent releases of the system
- full-time engagement of customer
- pair programming, collective ownership of the code, sustainable development
- regular system releases, test-first development, continuous integration
- constant refactoring, simplest thing that can work



Source: Sommerville: Software Engineering, 8th edition, 2007

Driving a car is not about pointing the car in one direction and holding to it; driving is about making lots of little course corrections.

"Do the simplest thing that could possibly work"

A well-known experience: The customer and the developer sit in a boat in the ocean and are afraid of each other

Customer fears	Developer fears
They won't get what they asked for	They won't be given clear definitions of what needs to be done
They must surrender the control of their careers to techies who don't care	They will be given responsibility without authority
They'll pay too much for too little	They will be told to do things that don't make sense
They won't know what is going on (the plans they see will be fairy tales)	They'll have to sacrifice quality for deadlines

# Result: a lot of energy goes into protective measures and politics instead of success

You have the right to an overall plan	To steer a project, you need to know what can be accomplished within time and budget
You have the right to get the most possible value out of every programming week	The most valuable things are worked on first.
You have the right to see progress in a running system.	Only a running system can give exact information about project state
You have the right to change your mind, to substitute functionality and to change priorities without exorbitant costs.	Market and business requirements change. We have to allow change.
You have the right to be informed about schedule changes, in time to choose how to reduce the scope to restore the original date.	XP works to be sure everyone knows just what is really happening.

You have the right to know what is needed, with clear declarations of priority.	Tight communication with the customer. Customer directs by value.
You have the right to produce quality work all the time.	Unit Tests and Refactoring help to keep the code clean
You have the right to ask for and receive help from peers, managers, and customers	No one can ever refuse help to a team member
You have the right to make and update your own estimates.	Programmers know best how long it is going to take them
You have the right to accept your responsibilities instead having them assigned to you	We work most effectively when we have accepted our responsibilities instead of having them thrust upon us

Customer makes business decisions

Developers make technical decisions

<b>Business Decisions</b>	<b>Technical Decisions</b>
Scope	Estimates
Dates of the releases	Dates within an iteration
Priority	Team velocity
	Warnings about technical risks

The Customer owns "what you get" while the Developers own "what it costs".

Describing XP



#### Basic XP Values

#### Communication

communicate problems&solutions, teamwork

Simplicity

- eliminate wasted complexity

Feedback

- change creates the need for feedback

Courage

- effective action in the face of fear

Respect

- care about you, the team, and the project

#### Principles

Humanity, Economics, Mutual Benefit, Self-Similarity, Improvement, Diversity, Reflection, Flow, Opportunity, Redudancy, Failure, Quality, Baby Steps, Accepted Responsibility

Will not detail them -- they govern what the practices tend to accomplish

So, on to the practices!

#### **Primary Practices**

Sit Together

Whole Team

Informative Workspace

Energized Work

Pair Programming

Stories

Weekly Cycle

Quarterly Cycle Slack Ten Minute Build Continuous Integration Test-First Programming Incremental Design

#### Stories

plan using units of customer-visible functionality



#### Another example

Ca- forchase facking passes. Cats \_ THE R. P. LEWIS CO., LANSING MICH. M commencements of the second commencement and the accurate service the second s and an experimental state of the second state a second design of the second the second s THE R. P. LEWIS CO., LANSING, MICH. & CO., M 11

Students can purchase monthly parking passes online.

Parking passes can be paid via credit cards.

Parking passes can be paid via PayPal <sup>™</sup>.

Professors can input student marks.

Students can obtain their current seminar schedule.

Students can only enroll in seminars for which they have prerequisites.

Transcripts will be available online via a standard browser.

Develop in an open space big enough for the team



#### Informative Workspace

#### Workspace = about your work

- 15 seconds to convey how project is going
- shows important, active information
- drinks & snacks available, and clean



## Pair Programming

Write all production programs with two people sitting at one machine

- make enough room, move keyboard and mouse
- Pair programmers:
  - keep each other on task
  - brainstorm refinements to the system
  - clarify ideas
  - take initiative when partner is stuck (less frustration)
  - hold each other accountable to practices

# Pair programming and privacy

Sometimes you might need some privacy

- then go work alone
- come back with the idea (NOT the code)
  - quickly reimplemented with two
  - benefits the whole team, not you alone

#### Pair Programming

#### Rotate pairs frequently

- every couple of hours, at natural breaks in development
- with a timer, every 60 minutes (or 30 minutes for difficult problems)

# Pair Programming and Personal Space

- Not everybody likes to sit close!
- Observe personal hygiene and health
- Sexual feelings are not in best interest of the team
  - even when mutual
- When uncomfortable pairing with somebody, talk about it with someone safe
  - chances are that you are not the only one
  - everybody needs to feel comfortable

### Weekly Cycle

Plan work one week at a time.

Do this on a meeting at the begin of each week

- Review progress.
- Let customers pick a week's worth of stories to implement this week.
- Break the stories into tasks. Team members sign up for tasks and estimate them.

Start writing tests that will run when the stories are completed

#### Ten-Minute Build

Automatically build the whole system and run all of the tests in ten minutes

- longer: will not be used (and errors result)
- shorter: not enough time to drink coffee

Note: if it takes longer than 10 minutes:

- maybe only rebuild changed part or test changes
- But: introduces errors. Only do this when necessary

Lowers stress: "Did we make a mistake? Let's see."

#### **Continuous Integration**

#### Team Programming = Divide, Conquer, Integrate

Integrate and test changes after no more than a couple of hours

- integration typically takes long
- when done at the end, risks the whole project when integration problems are discovered
- the longer you wait, the more it costs and the more unpredictable it becomes

# Using Continuous Integration

#### Synchronous

- After a task is finished, you integrate and run the tests
- Immediate feedback for you and your partner

#### Asynchronous

- After submitting changes, the build system notices something new, builds and tests the system, and gives feedback by mail, notification, etc.
- Feedback typically comes when a new task is started
- Pair programmers might have been switched already

# Test-first Programming

Write a failing automated test before changing code

Addresses many problems:

- Scope creep: focus coding by what the code should do, not on the "just in case" code
- Coupling and cohesion: If it's hard to write a test, there is a design problem (not a testing problem)
- Trust: clean working code + automated tests
- Rhythm: gives focus on what to do next
  - efficient rhythm: test, code, refactor, test, ...

#### Incremental Design

Invest in the design of the system every day. Strive to the design of the system an excellent fit for the needs of the system that day

- Completely opposite to lots of other practices
  - Waterfall and similar approaches
- Can work with XP because of the other practices
  - Automated tests, continuous integration, ...
- Note: you need to invest in design!
  - not just implement story after story after story...

#### **Corollary Practices**

Real Customer Involvement Incremental Deployment

**Team Continuity** 

Shrinking teams

**Root-Cause Analysis** 

Shared Code

Code and Tests Single Code Base Daily Deployment Negotiated Scope Contract Pay-Per-Use

#### Initiation

- User Stories
- Release Planning

Release (each Release is typically 1 -6 months)

- Iteration 1 (typically 1 -3 weeks)
- Iteration 2
- :
- Iteration n

#### Advantages

- works well for small teams
- low process overhead, lean & mean

Potential pitfalls

- no documented compromises of user conflicts
- lack of an overall design specification or document
- can be hard to fit in organizations/workflows

#### Conclusion

#### A software development methodology is

- a set of partially ordered steps
- to build, deploy, maintain, ... software

#### Many methodologies exist

- each with trade-offs
- pick the one according to your needs
  - project size
  - project partners
  - development team(s)
  - outside constraints (legislation, domain constraints, ...)

#### References

Craig Larman, Applying UML and Patterns – An Introduction to Object-Oriented Analysis and Design and Iterative Development (3rd ed.), Prentice Hall, 2005.

Kent Beck, Extreme Programming Explained: Embrace Change (2nd ed.), 2004.

http://c2.com/cgi/wiki?ExtremeProgramming

#### License: Creative Commons 4.0

#### You are free to:

- Share copy and redistribute the material in any medium or format
- Adapt remix, transform, and build upon the material

for any purpose, even commercially.

The licensor cannot revoke these freedoms as long as you follow the license terms.

#### Under the following terms:



Attribution — You must give <u>appropriate credit</u>, provide a link to the license, and <u>indicate</u> <u>if changes were made</u>. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.



**ShareAlike** — If you remix, transform, or build upon the material, you must distribute your contributions under the **<u>same license</u>** as the original.

**No additional restrictions** — You may not apply legal terms or **technological measures** that legally restrict others from doing anything the license permits.