Agile Software Development (DIT191 / EDA397)
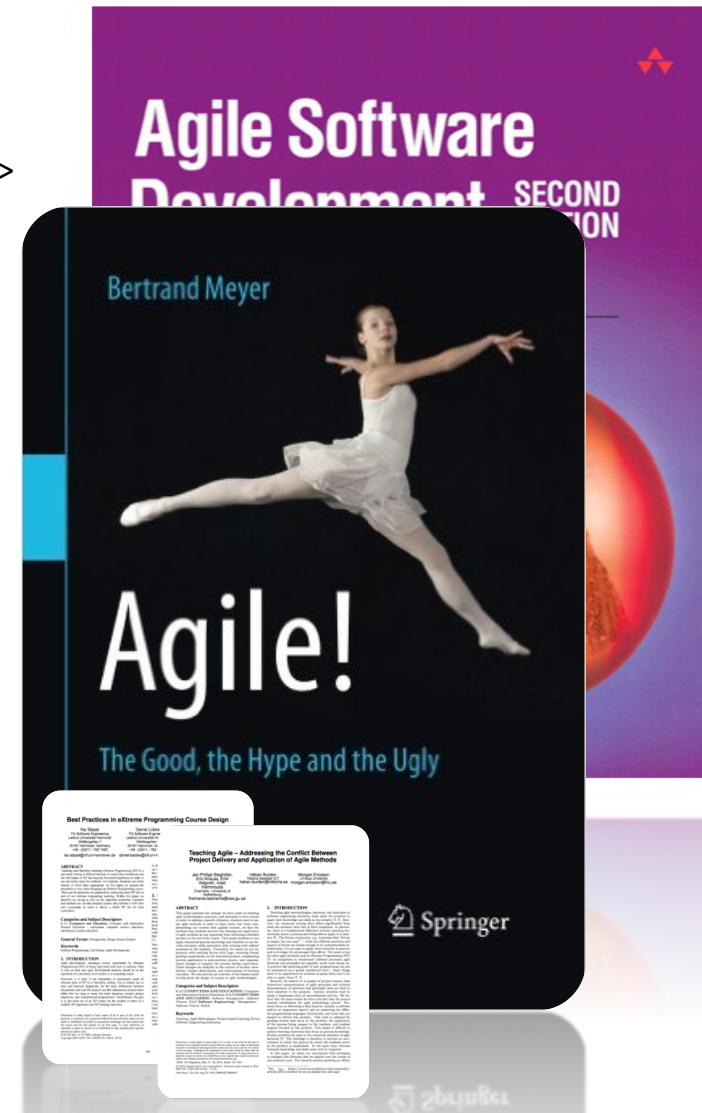
Spring 2017
Eric Knauss
eric.knauss@cse.gu.se
https://github.com/oerich/EDA397

Course setup

- Teaching Assistants
  - Terese Besker <besker@chalmers.se>
  - Magnus Ågren <magnus.agren@chalmers.se>

- Course representatives

- In parallel:
  - PhD course

- Course Material
  - http://oerich.github.io/EDA397/
  - Meyer, B., (2014) Agile! The Good, the Hype and the Ugly
  - Papers

Course setup (Practical details)

- ## Schedule
  - 0-3 lectures per week
  - 0-2 workshops per week
  - =3 scheduled activities per week
    - Even if there is no lecture, we will be available and you can (should!) use the rooms/time to work!

- ## Examination
  - Project (teams)
    - Final product
    - Artifacts
    - Experience / Post-Mortem report
  - Written exam

Examination

- Written exam, individual, 3.0 credits
  - 60 points, 24 required to pass
- Grades
  - Chalmers:
    - X < 24 → Fail,
    - 24 ≤ X < 36 → 3,
    - 36 ≤ X < 48 → 4,
    - 48 ≤ X → 5
  - GU
    - X < 24 → Fail,
    - 24 ≤ X < 48 → G,
    - 48 ≤ X → VG
- Project, 4.5 credits
  - Grades: Fail/Pass
    based on project participation and group report

Project

- Develop software for a customer

- Work in predetermined teams
  - https://github.com/oerich/EDA397/wiki/Groupings-of-Teams
  - You will be assigned teams
  - Focus on agile practices and methodology
  - You meet with the customer this Thursday
    - and get all the details!

- We strive to create a realistic scenario/ environment
  - We rely on a number of real-world services and tools, e.g.
  - Android (SDK), GitHub, Trello, (perhaps Jenkins, Gerrit, SonarCube…)

Course setup

- Three sprints

- Sprint 1: Getting started

- Sprint 2: Getting work done

- Sprint 3: Theory and advanced concepts

Course Objectives

Knowledge and understanding	Skills and ability	Judgement and approach
Compare agile and traditional softw. dev,	Forming a team organically	Explain: people/commun. centric dev.
Relate lean and agile development	Collaborate in small software dev. teams	Apply fact: people drive project success
Contrast different agile methodologies	Interact and show progress continuously	Describe: No single methodology fits all
Use the agile manifest and its accompanying principles	Develop SW using small and frequent iterations	Discuss: methodology needs to adopt to culture
Discuss what is different when leading an agile team	Use test-driven dev. and automated tests	
	Refactor a program/design	
	Be member of agile team	
	Incremental planning using user stories	

Sprint 1

Sprint 2

Sprint 3

Changes since last year

- More choice in project part

- Different course book
  - More hands-on, more critical

- Updated slides (ongoing)

- Updated reading list (ongoing)
  - Nothing will be added!
  - We will indicate which of the papers will not be on the exam

- Much better timeline
  - First iteration done before Easter break!

Sprint 1: Getting started

http://commons.wikimedia.org/wiki/File:Sprint_01.jpg

What is agility in Software Development?

Agile: An Overview

http://mediagallery.usatoday.com/New+Flame

Motivation

- What is the "Software crisis"?

  – Software development inefficient
  – Software does not meet requirements
  – Projects over time/budget
  – Projects were unmanageable and software unmaintainable

- What can be done about it?

→ Software Engineering
  – Application of engineering to Software
  – Systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software
  – Assure quality of process and product

?
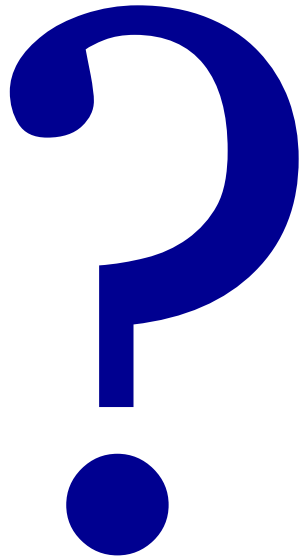
- Do you know examples of

  – Systematic, disciplined, quantifiable approaches to the development, operation, and maintenance of software?

  – Applying engineering to software?

Where are we coming from?

Requirements

Design

Programming

Test

http://commons.wikimedia.org/wiki/File:Waterfall_katoomba.JPG

# Systematic sequential development

- Requirements
- Design
- Programming
- Test

- Advantages
  – Simple
  – Controllable
  – Cost efficient

- Problems
  – Time-to-market
  – What about change?

Towards concurrent development

- Requirements
- Design
- Programming
- Test
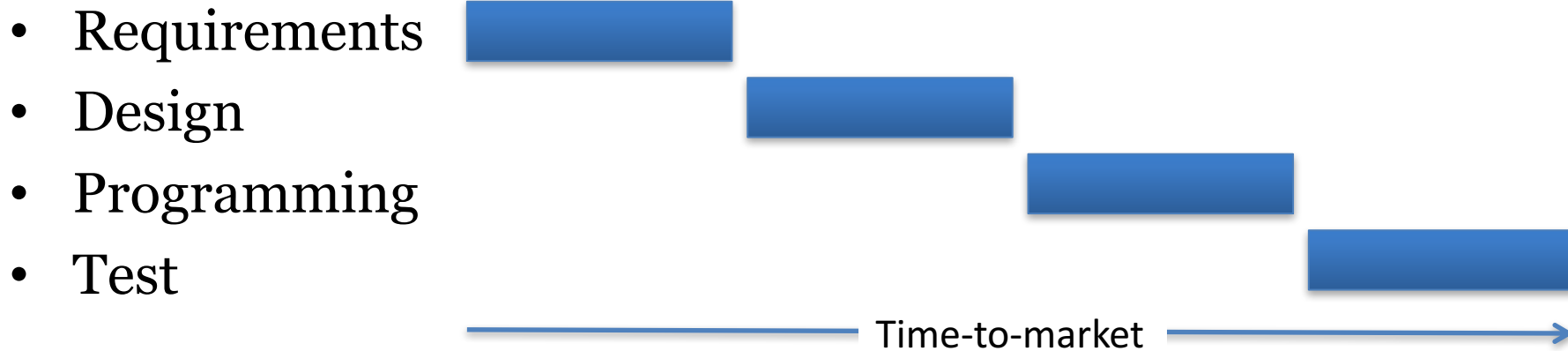
Time-to-market

What can we do if time to market and robustness against late changes are more important than cost-efficiency?

- Requirements
- Design
- Programming
- Test

Time-to-market

?
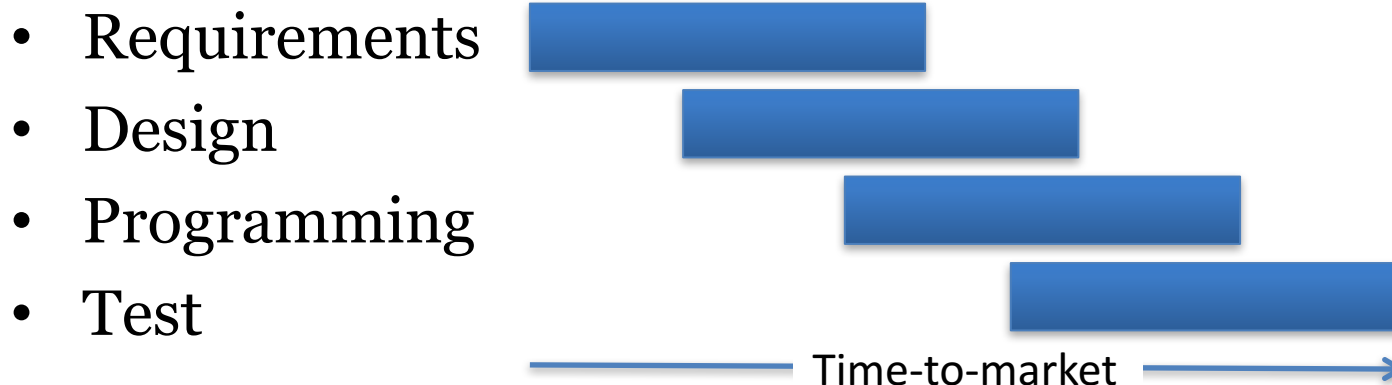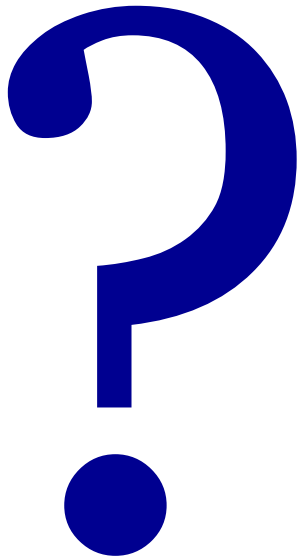
- What is the consequence of concurrent development?
  - (hint: why are concurrent tasks depicted longer than sequential tasks?)

- What has this to do with agile?

- What do you know about agile?

Agile Manifesto

Manifesto for Agile Software Development

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.

http://agilemanifesto.org

- Began as a provocation: Plan-driven development did not safe the Software world…
- Now a very serious movement, well adapted in industry.
- There are a couple of established agile methods: How to integrate these values in everyday software development

What is agile? What is not?

- Agile – a compendium of ideas
  - Applied by number of methods
    (incl. XP, Scrum, Kanban, Lean Software Development)

- Core characteristics defined through
  - **Values**: General assumptions framing the agile view of the world
  - **Principles**: Core agile rules, organizational and technical
  - **Roles**: responsibilities and privileges of the various actors in an agile process
  - **Practices**: specific activities practiced by agile teams
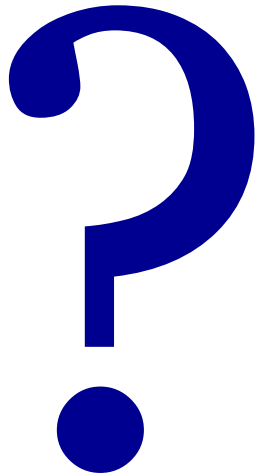 - **Artifacts**: tools, both virtual and material, that support the practices

[Mey2014]

Agile Values

1. Redefined roles for developers, managers, and customers
2. No "Big Upfront" steps
3. Iterative development
4. Limited, negotiated functionality
5. Focus on quality, understood as achieved through testing

[Mey2014]

?

- Can the following projects be agile?
  - App development
  - Online shop
  - Mission controller for Airplane
  - Controller for nuclear plant

- What is a "principle"?
  - Abstract
  - Falsifiable
  - Prescriptive

- Discuss the following agile principles.
  - Do they help to define what is agile and what not?
  - Any problems in the list?

#	Agile Principle	Comment
1	Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.	
2	Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.	
3	Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.	
4	Business people and developers must work together daily throughout the project.	
5	Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.	
6	The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.	
7	Working software is the primary measure of progress.	
8	Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.	
9	Continuous attention to technical excellence and good design enhances agility.	
10	Simplicity--the art of maximizing the amount of work not done--is essential.	
11	The best architectures, requirements, and designs emerge from self-organizing teams.	
12	At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.	

[Mey2014]

#	Agile Principle	Comment
1	Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.	
2	Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.	
3	Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.	
4	Business people and developers must work together daily throughout the project.	
5	Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.	
6	The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.	Practice
7	Working software is the primary measure of progress.	
8	Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.	
9	Continuous attention to technical excellence and good design enhances agility.	
10	Simplicity--the art of maximizing the amount of work not done--is essential.	
11	The best architectures, requirements, and designs emerge from self-organizing teams.	
12	At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.	Practice

[Mey2014]

#	Agile Principle	Comment
1	Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.	
2	Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.	
3	Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.	
4	Business people and developers must work together daily throughout the project.	
5	Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.	Platitude
6	The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.	Practice
7	Working software is the primary measure of progress.	
8	Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.	
9	Continuous attention to technical excellence and good design enhances agility.	Platitude
10	Simplicity--the art of maximizing the amount of work not done--is essential.	
11	The best architectures, requirements, and designs emerge from self-organizing teams.	
12	At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.	Practice

#	Agile Principle	Comment
1	Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.	
2	Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.	
3	Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.	
4	Business people and developers must work together daily throughout the project.	
5	Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.	Platitude
6	The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.	Practice
7	~~Working software is the primary measure of progress.~~	Assertion

"Use working software as the primary measure of progress."

| 8 | Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely. | |
| 9 | | Platitude |

"Seeking simplicity" = ok; "Maximizing work not done" = ok; but two principles

10	~~Simplicity--the art of maximizing the amount of work not done--is essential.~~	Assertion
11	The best architectures, requirements, and designs emerge from self-organizing teams.	
12	At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.	Practice

[Mey2014]

#	Agile Principle	Comment
1	Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.	Duplicate (3)
2	Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.	
3	Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.	Duplicate (3,7)
4	Business people and developers must work together daily throughout the project.	
5	Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.	Platitude
6	The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.	Practice
7	Working "Use working software as the primary measure of progress."	Assertion Dupl. (3)
8	Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.	
9	Continuous attention to technical excellence and good design enhances agility.	Platitude
10	"Seeking simplicity" = ok; "Maximizing work not done" = ok; but two principles	Assertion
11	The best architectures, requirements, and designs emerge from self-organizing teams.	
12	At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.	Practice

Missing: Testing

[Mey2014]

Agile Principles – Revised list
(according to [Mey2014])

Organizational

1. Put the customer at the center.

2. Let the team self-organize.

3. Work at a sustainable pace.

4. Develop minimal software:

    1. Produce minimal functionality.

    2. Produce only the product requested.

    3. Develop only code and tests.

5. Accept Change

Technical

1. Develop iteratively:

    1. Produce frequent working iterations.

    2. Freeze requirements during iterations.

2. Treat tests as a key resource:

    1. Do not start any new development until all tests pass.

    2. Test first.

3. Express requirements through scenarios.

Agile Roles

Role	Description
Manager	A supporting role ("A guru, not a nanny!")
Product Owner	Define and maintain the product backlog: The list of features. Before dev.: Explain them. After dev.: Accept/reject them
Team	Takes over manager responsibilities (incl: deciding what tasks to implement)
• Self-organized	Optimize intense collaborations within and across organizational boundaries to meet challenges as they arise.
• Cross-functional	Team includes all necessary skills to deliver a feature, independently of other teams.
Members vs. Observers	Members should dominate discussions, observers should offer opinions when asked (c.f. Chicken and Pigs)
Customer	Central. Constant interaction prescribed in XP, replaced through indirection (Product Owner) in Scrum.
Coach, Scrum-Master	Help sticking to agile values, principles, practices. Main question: dedicated role or not.

[Mey2014]

One Agile Method: Extreme Programming

- Resonates well with our course
  - Focus on developer

- 1st and 2nd Edition differ quite significantly
  - We stick to the first

- Today:
  - Most important practices for this week

- Homework:
  - Pick two of the practices, read up on them, prepare to explain them in 1-2min
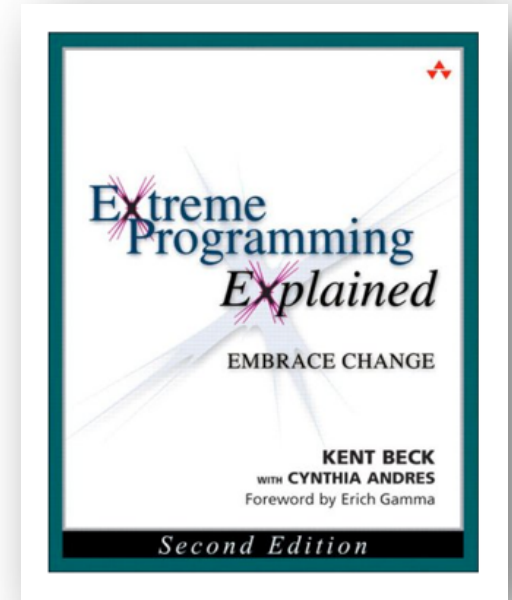
Agile Methods: eXtreme Programming (XP)

- Extreme programming:
  - An approach based on the development and delivery of very small increments of functionality

  - No fine grained process description, but 12 practices arranged around short development circles (4-6 weeks)

  - "Turn-to-ten" metaphor (refers to volume setting of older amplifiers):
    - Reviewing is good? → Review continuously: Pair Programming
    - Early Tests are good? → Write tests before code: Test-First
    - Customer interaction is good? → Have Onsite-Customer
    - …

XP: Planning/Feedback Loops

Release Plan → Months → Iteration Plan → Weeks → Acceptance Test → Days → Stand Up Meeting → One day → Pair Negotiation → Hours → Unit Test → Minutes → Pair Programming → Seconds → Code
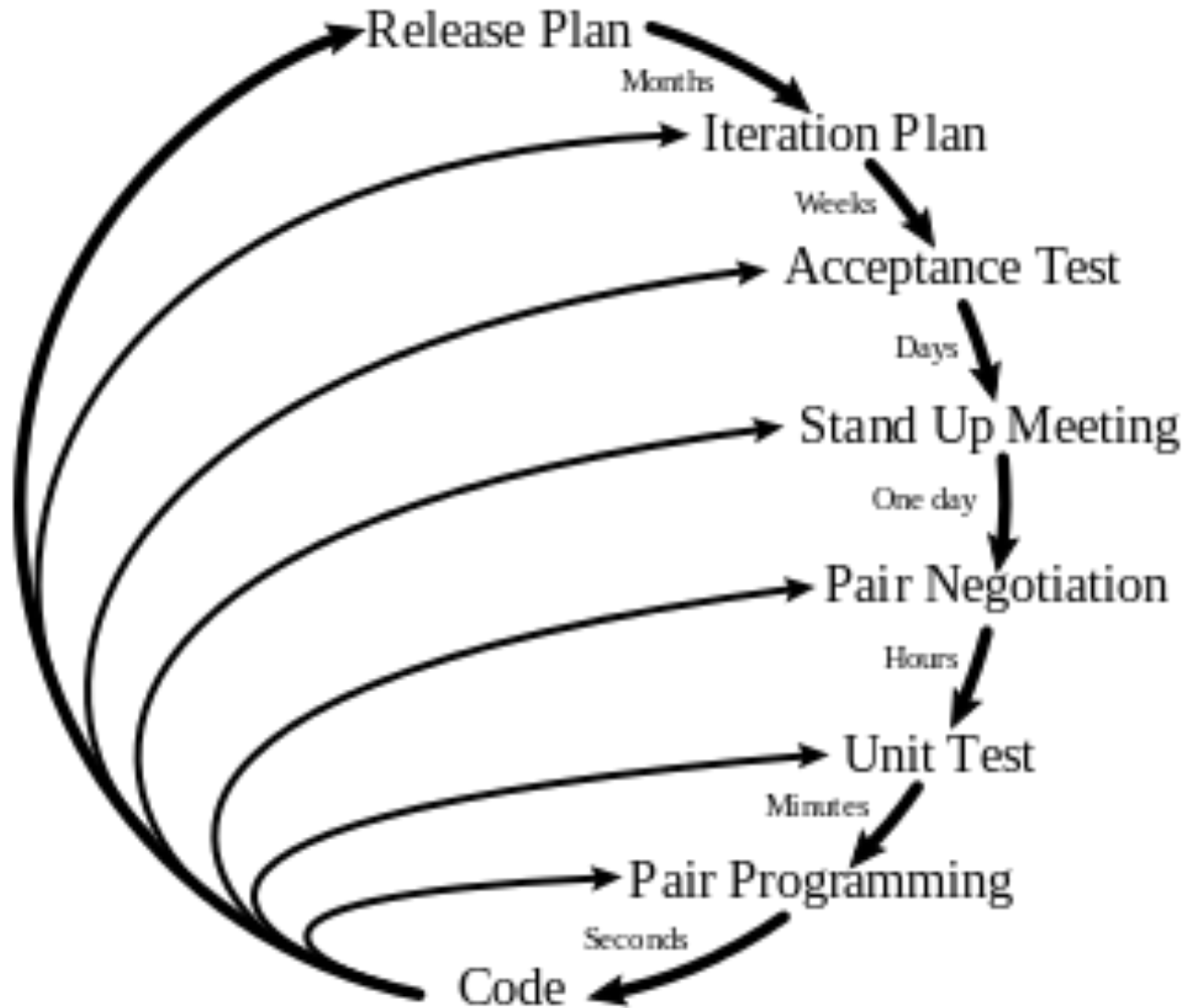
http://en.wikipedia.org/wiki/File:Extreme_Programming.svg

XP Practices

Pair Programming

Onsite-Customer

Test-first

Planning game

Feedback

Continuous process

Small releases

Continuous integration

Refactoring

Shared understanding

Coding standards

Collective Code Ownership

System metaphor

Simple design

Sustainable pace

Programmer welfare
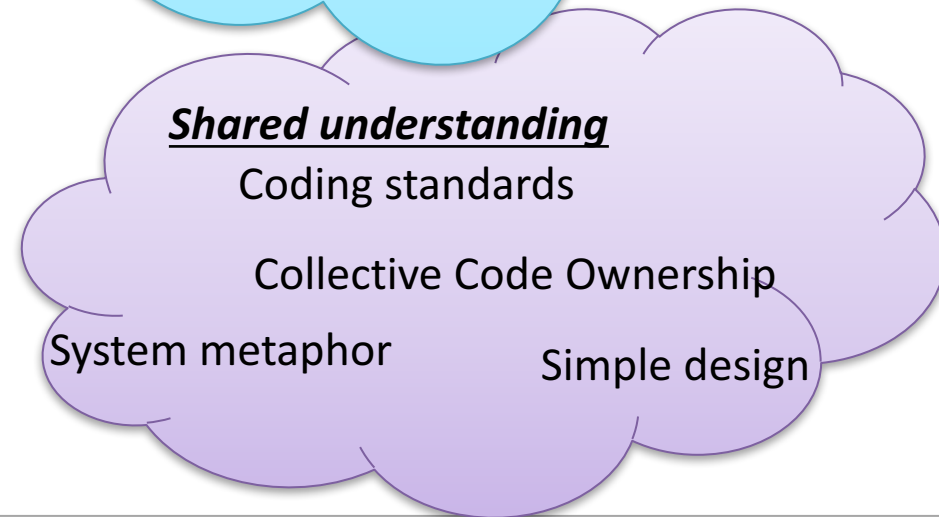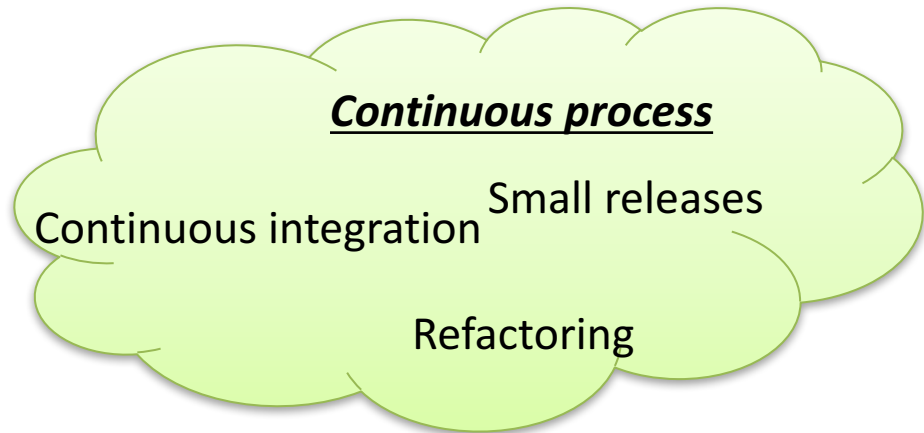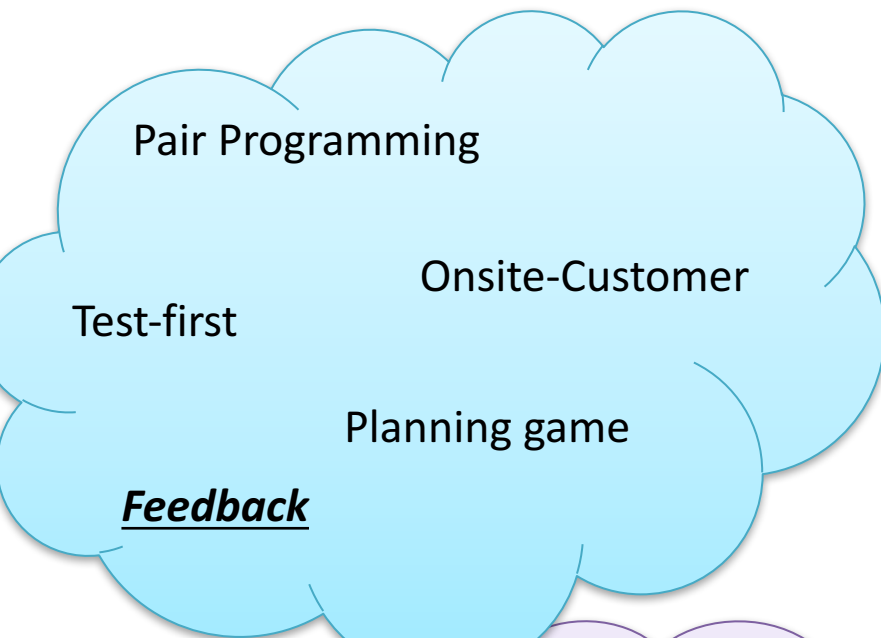
Agile Principles and Practices
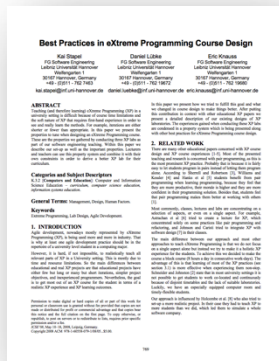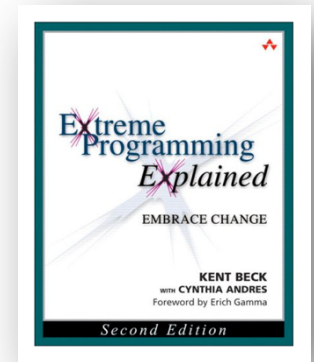
- ## Goal: Try them out in your project!

	Mandatory	Optional	Comment
Planning Game	1	0	*Make the most out of it. Get Emil's Priorities based on your effort estimation. Employ customer proxy*
Small Releases	1	0	
Metaphor	0	1	*Try it out! But we will not check whether it works.*
Simple Design	1	0	
Test-First	1	0	*But only where it makes sense. Have a good rationale!*
Refactoring	1	0	
Pair Programming	0,5	0,5	*Try it out. Don't necessarily do it all the time.*
Collective Codeownership		0	*Everybody should know about the code. Some parts more than others*
Continuous Integration	1	0	
Sustainable Pace	1	0	*But also not too slow!*
Onsite Customer	0,5	0,5	*Have a customer proxy*
Coding standards	1	0	*Decide on them and try to have tool support*

On-Site Customer

- ## Real customer in the room
  - Answers all questions now (…and can revise answer later)
  - Customer proxy
- ## *Answer now* more important than *answer correct*
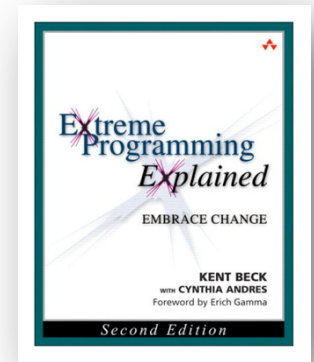
"Students have to interact with a designated On-Site Customer who is available full-time to answer questions."
- (+) Block course, customer interest

Planning Game

- ## Business people need to decide
  - Scope, Priority, Composition of release, dates of release

- ## Technical people need to decide
  - Estimates, Consequences, Process, Detailed scheduling

[Bec1999]
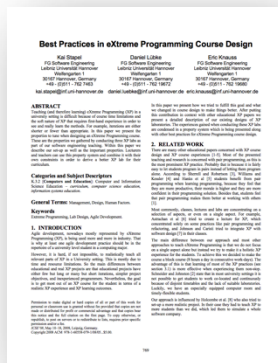
[SLK2008]

"Students learn how to divide requirements into User Stories and how to prioritize and estimate the costs of these stories. While such tasks seem to be easy in theory, dealing with dependencies in the planning game is normally a challenge for inexperienced developers like students."

- (+) More iterations, small teams, customer interested, technical support, progress feedback
- (-) Longer iterations

Continuous Integration

- Integrate and test code every few hours (1 day at the most)

- Dedicated machine helps
  - If Machine is free: pair sits down, integrates their changes, tests, and does not leave before 100% of tests run
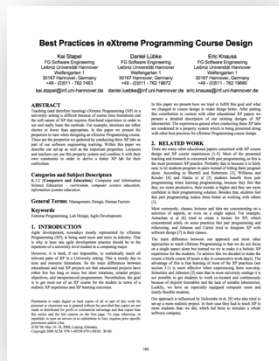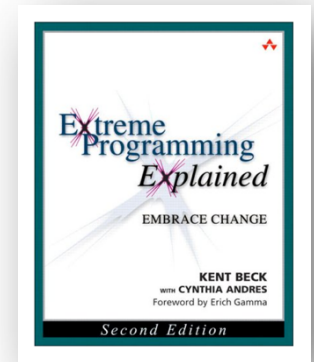
"To counter conflicting updates to the code base, students learn to integrate and build the software frequently."

- (+) Block course, longer iterations, more iterations, small team size

- (discovered later: technical feedback)

Simple Design

- ## The right design at any given time
  - Runs all the tests
  - Has no duplicated logic
  - States every intention important to the programmers
  - Has the fewest possible classes and methods
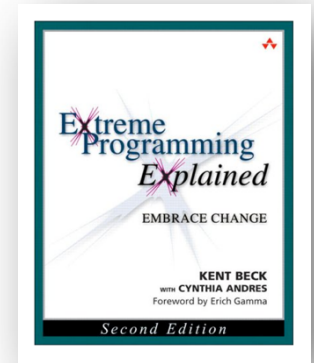
"Students learn the benefits of simple software design which improves their ability to change the system quickly and accommodate it to changing requirements."

- (+) More iterations, technical support

Small releases

- **Every release**
  - ... should be as small as possible
  - ... should contain the most valuable business requirements
  - ... has to make sense as a whole
  - ... should be delivered every 4-8 weeks (rather than 6-12 month)
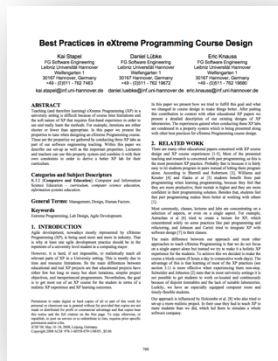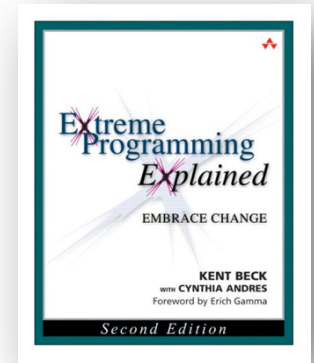
"Students learn the benefits of small releases that already offer value to the customer and how to technically put a system into production including packaging."
- (+) More iterations, progress feedback

Metaphor

- ## Examples
  - Naïve: "Contract management system deals with contracts, customers, and endorsements"
  - "Computer should appear as a desktop"
  - "Pension calculation is a spreadsheet"
  - Align team thinking
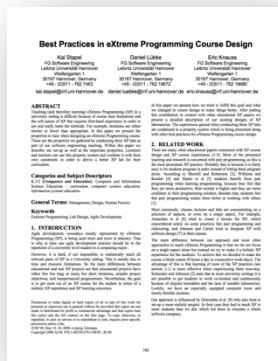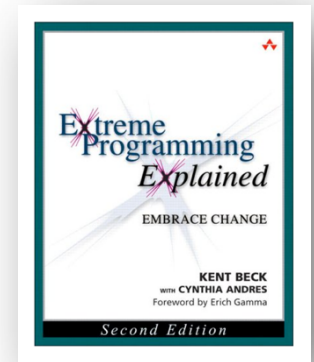
"Students learn how to develop a metaphor that helps every team member to better understand how the whole system works."
- (+) Technical support, technical feedback

Testing

- Any feature without automated test does not exist
  - Don't write a test for every method
  - Write a test for every productive method that could possibly break
  - "Program becomes more and more confident over time"
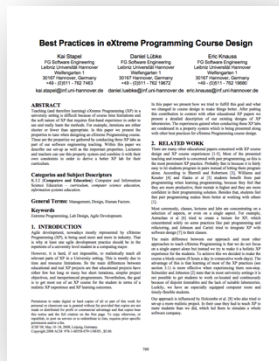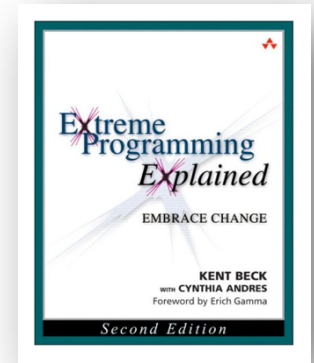
"Students learn how to use unit test frameworks and the test-first approach to build high quality software and to recognize the advantages of well-tested code when making changes."

- (+) More iterations, **technical support, technical feedback**

Refactoring

- Is there a way of changing the program to make it easier to add a new feature?

- After adding the feature: Can we simplify the design?

- Important investment!!!

"Students learn to refactor the software to remove duplication, improve communication and simplify the code base. Especially refactoring large systems can be troublesome and is a worthy experience that can only be made in long lasting projects."
- (+) More iterations, customer interest, technical support

Pair Programming

- Driver: Has the keyboard, writes the code
- Navigator: Thinking strategically
 - Will this work? Which test cases might not work? Can we simplify?
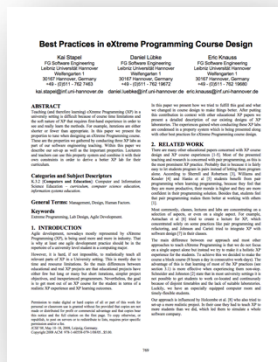
"Students learn and experience the principles of pair programming, the advantages of writing software with a partner and the social challenges that are associated."

- (+) Block course, **small team size**, technical support
- (-) Long iterations

Collective Code Ownership

- "Anybody who sees an opportunity to add value to any portion of the code is required to do so at any time."
  - No code ownership → Chaos
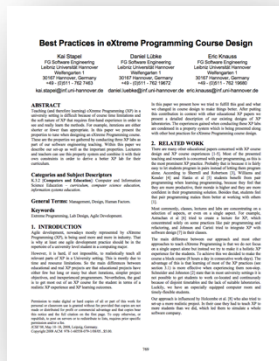 - Individual code ownership → Stable but slow
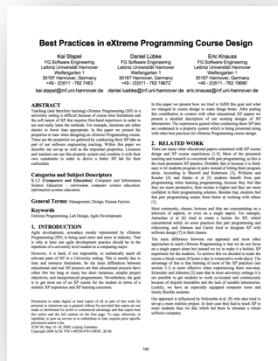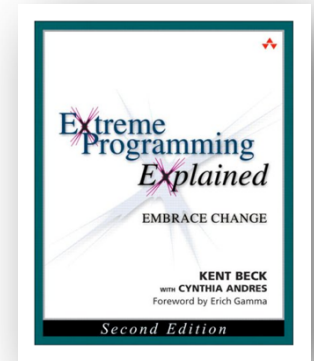
"Students get to know the advantages of collective code ownership and the challenges that arise with parallel updates and changes to their own code by other team members. "

- (+) Block course, small team size

Sustainable pace (aka 40h week)

- Be fresh every morning, tired every night
- One week of overtime must not be followed by another one
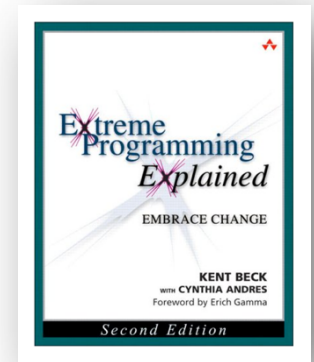
"In contrast to normal life in university, students experience to work continuously for 40-hours per week in a designated team room."

- (+) **Block course**

Coding Standards

- Swapping partners, changing concurrently all parts of the code...

- Your code should better look consistently!
  - Once and only once rule
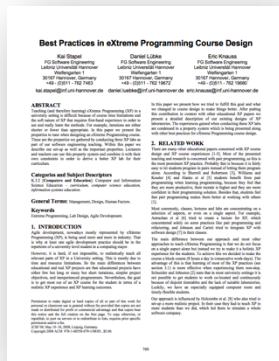 - Emphasize communication
  - Adopted by whole team
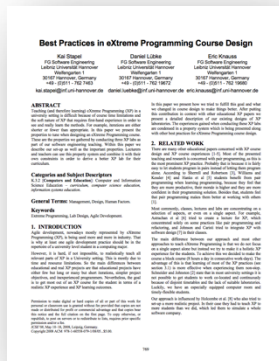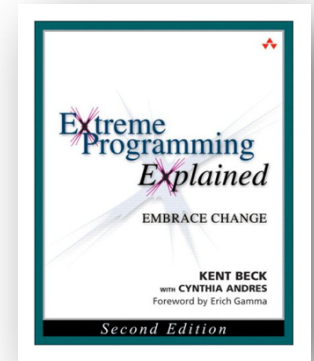
"Students experience the importance of uniform coding conventions throughout the team especially when combined with collective code ownership."

References
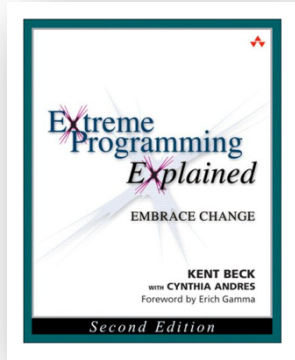
[Bec1999]   Kent Beck, Extreme Programming Explained, Addison-Wesley, 2ed, 2000

[SLK2008]   Kai Stapel, Daniel Lübke, Eric Knauss: Best Practices in eXtreme Programming Course Design. In: *Proceedings of the 30th International Conference on Software Engineering (ICSE '08), Leibzig, Germany, pg.* 769-776, **2008**