Agile Development Processes (EDA397/DIT191)

Erik Knauss (Examiner)
Emil Alégroth (Teacher Assistant)

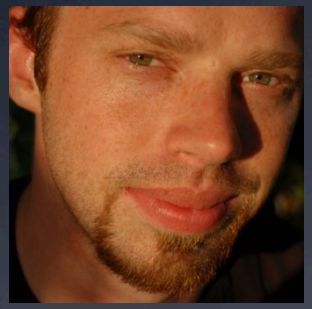
Introductions

Eric Knauss

- Currently on a conference. Will introduce himself when he gets back!

- Research:
  - Requirements Engineering
  - Agile methods
  - RE and Project Management
  - Experience and Knowledge Management

Introductions



Emil Alégroth

- MSc 2010 (Software Engineering)
- Phd Candidate 2011
- Licenciate degree 2013
- Planned PhD September 2015
- Research:
  - Automated testing
  - Model based testing
  - Alignment between requirements and tests
- Agile software development course
  - Teacher assistant in the course since 2011!

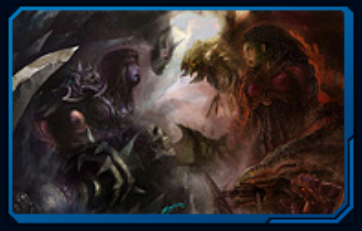
Revolution Overdrive: Songs of Liberty	26.11.10
Starcraft II: Wings of Liberty (Soundtrack)	27.07.10

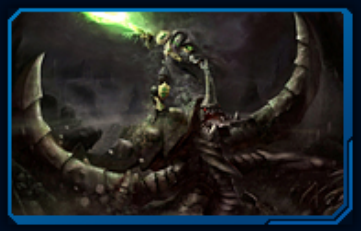
Zerg Creep Tumor	13/07/10
Terran Scanner Sweep	13/07/10
Battle.net UI Klaxon Alert	13/07/10

FAN ART (210)

All Fan Art >


Date Added: 26/02/13


Date Added: 26/02/13


Date Added: 08/01/13


Date Added: 08/01/13

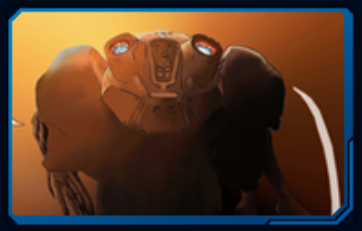
COMICS (27)

All Comics >

Date Added: 14/01/13

Date Added: 22/10/12

Date Added: 17/10/12

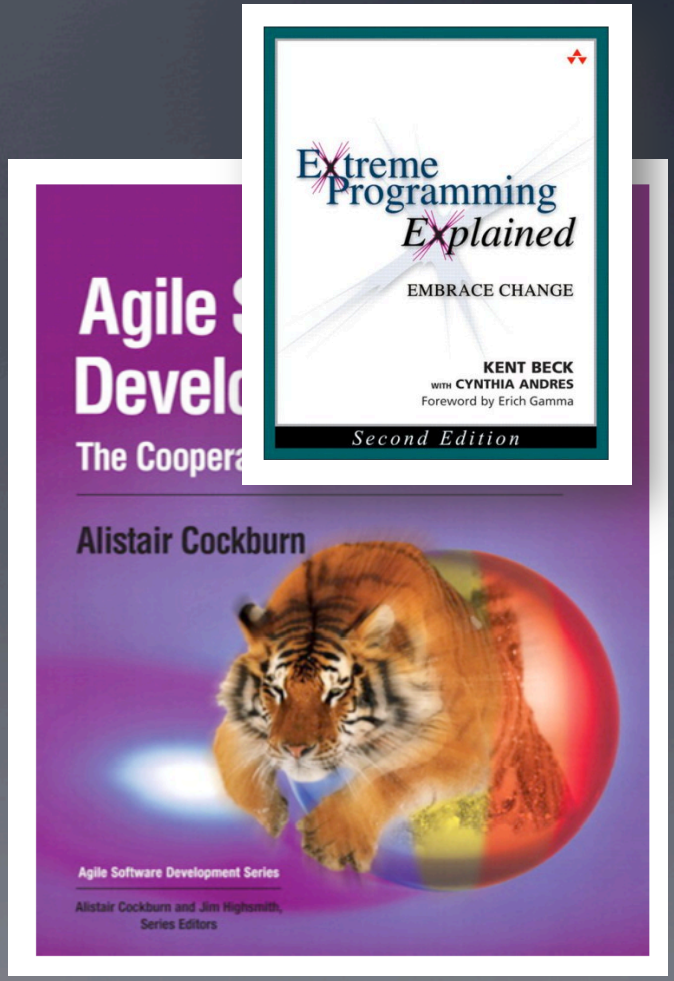

Date Added: 10/10/12

General information

- If you have any questions or comments send an e-mail
  - Emil.alegroth@chalmers.se
  - I answer as quickly as I can, but since I am busy it might take a day or two!
- You can also visit my office on the 4th floor in house Jupiter
  - Time constraints.
- Tell us ALL your problems with the course, fellow students, etc. But do so as quickly as possible!

Course information

- Course representatives
  - Christos Charalmpous
  - Markus Ekström
  - Linus Lindgren
  - Jing Wang
- We need a GU student to volunteer!
- Course Materials
  - http://oerich.github.io/EDA397/
  - Cockburn,A., (2009) Agile Software Development, 2ed
  - Papers



Course Webpage

http://oerich.github.io/EDA397/

Course setup

- 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 – You need to deliver something
    - Artifacts
    - Experience / Post-Mortem report
  - Written exam

Project

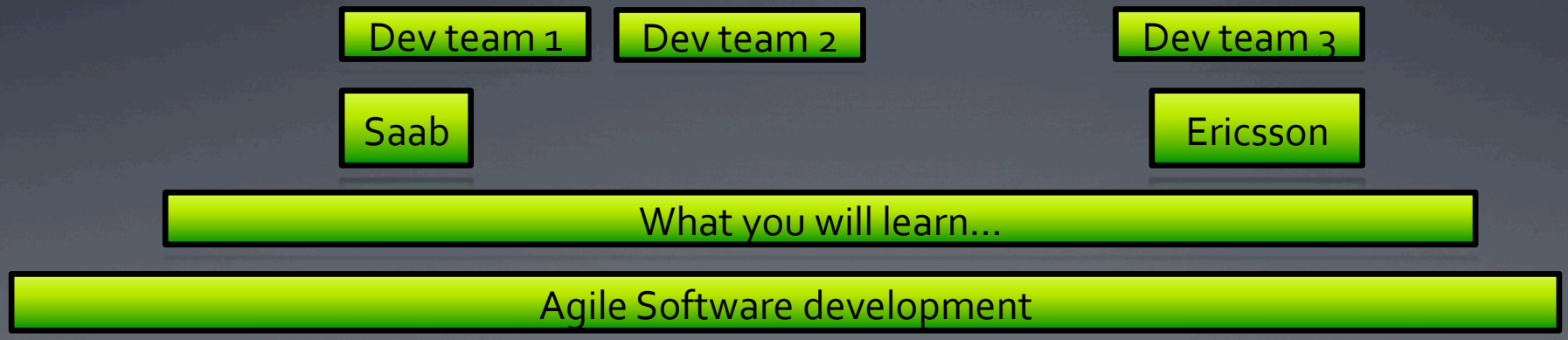
- Develop an Android app for a customer
- Work in predetermined teams
  - You will be assigned teams
  - We start with eXtreme programming
    - Then we add some other practices
  - You meet with the customer during course week 2
    - and get all the requirements!
- We strive to create a realistic scenario/ environment
  - We rely on a number of real-world services and tools, e.g.
  - Android (SDK) GitHub / BitBucket ..

Teams

- Teams will be randomly selected: NO EXCEPTIONS!
- Sign paper that has been sent around the room
  - Name
  - Personal number / Social security number
  - E-mail address
  - Development experience
- Thursday: Lecture where you will be assigned groups
  - Preferably meet your group members!
- Friday: Get started in your groups

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	


Dev team 1
Dev team 2
Dev team 3
Saab
Ericsson
What you will learn…
Agile Software development

Examination

- Written exam, individual, 3.0 credits
  - 60 points, 24 required to pass
- Grades
  - Chalmers:
    - [0-49%] → Fail,
    - [50-64%] → 3,
    - [65-79%] → 4,
    - [80-100%] → 5
  - GU
    - [0-49%] → Fail,
    - [50-79%] → G,
    - [80-100%] → VG

What is agility in Software Development?

Some quick questions

- Do you know how to write software (Java, C, C++, etc)?
- Have you been in a student project and written software?
- Have you worked as a software developer?
- Have you worked according to a software development process?
- Have you worked according to an agile software development process?

Taxonomy

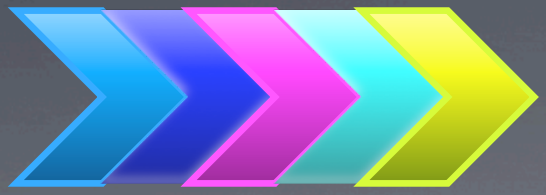
Product

The software system we are developing, e.g. Spotify, Photoshop, Facebook

Project

A limited time activity when we add new, or change, functionality to a product. Results in a new or changed product.

Process



The way we work during the project to add or change the product's functionality. Consists of a set of activities also referred to as practices.

Practice

A specific activity performed in a process, e.g. requirement elicitation, pair-programming, refactoring, system testing, etc.

All together

We develop a Product in Project(s) using a Process that consists of Practices!

Often in iterations!

OBS! JUST EXAMPLES!

Development in teams

	10.000+ LOC	Developer
	50.000+ LOC	Developers
 x5	100.000+ LOC	Dev/Arch/Test
 x50	1.000.000+ LOC	RE/Dev/Arch/Test
 X500+	Many million LOC	Massive organ.

Organizations need processes

Motivation for Software engineering

- 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

Traditional Software development

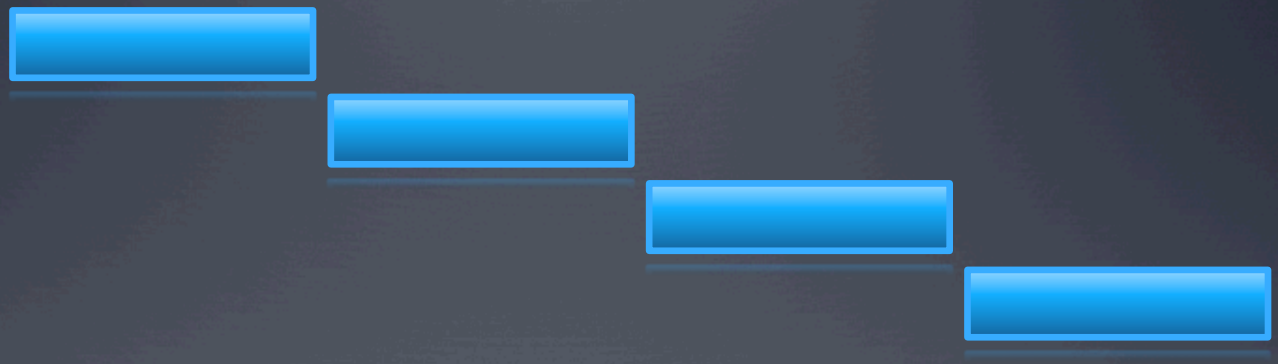
Interviews with customer
Write requirements
Turn requirements into tasks
Model the system

Write code

Unit testing
Integration testing
System Testing
Acceptance Testing

Systematic sequential development

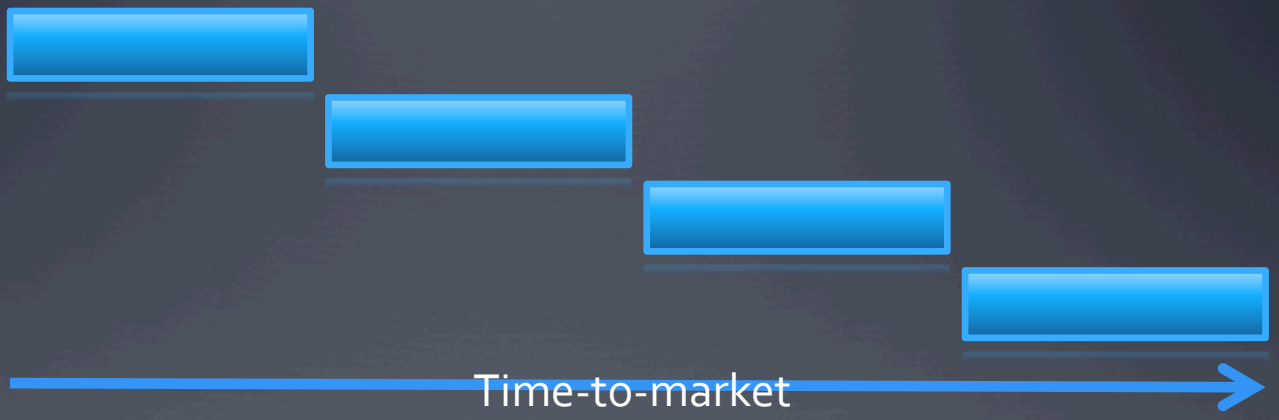
- Requirements
- Design
- Programming
- Test



- Advantages
  - Simple
  - Controllable
  - Cost efficient
- Problems
  - Time-to-market
  - What about change?

Towards concurrent development

- Requirements
- Design
- Programming
- Test



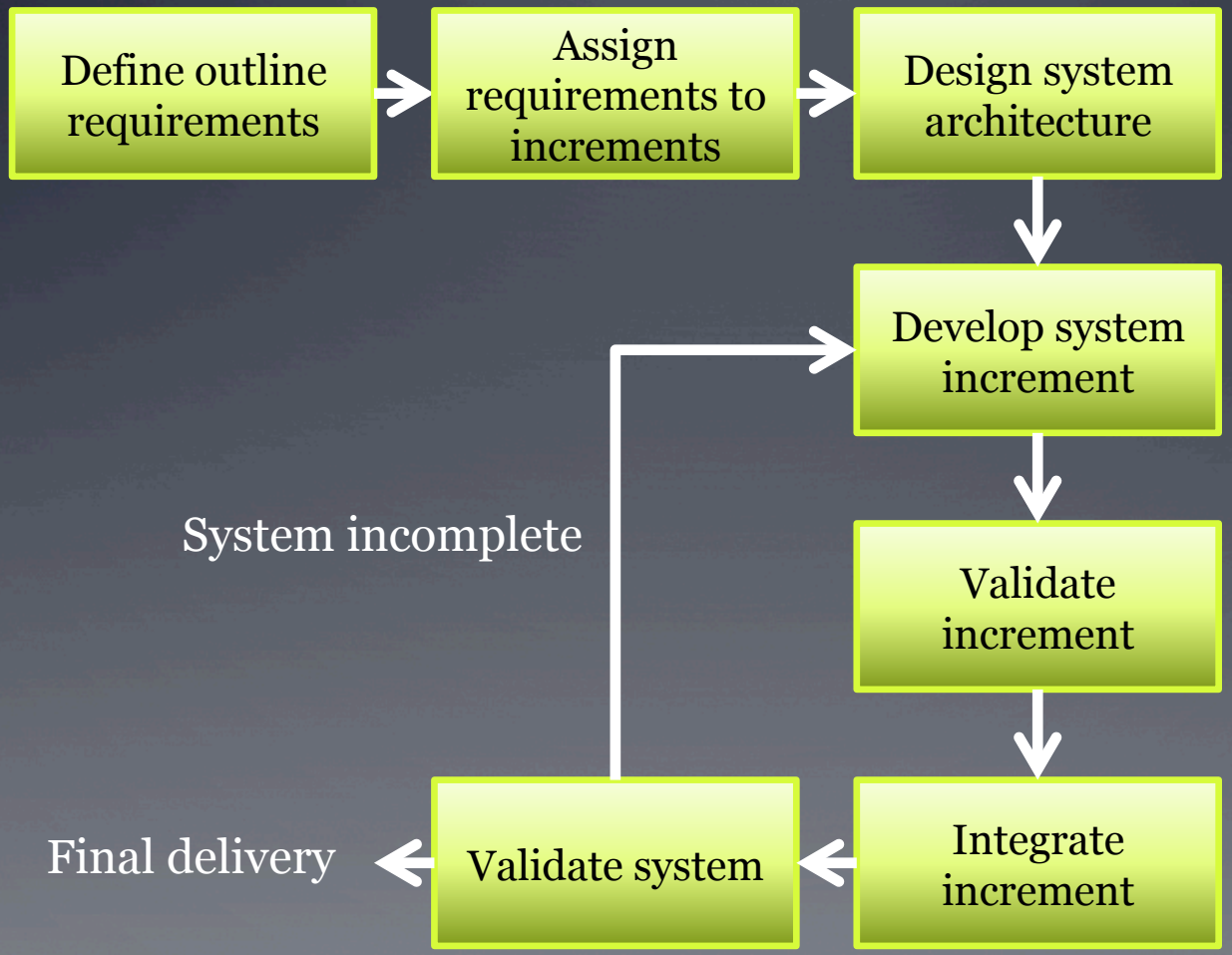
What can we do if time to market and robustness against late changes are more important than cost-efficiency?

- Requirements
- Design
- Programming
- Test

Iteration Models

- System requirements *always* evolve during a project
- Iterations are part of larger development projects
- Iterations can be applied to any generic development process model

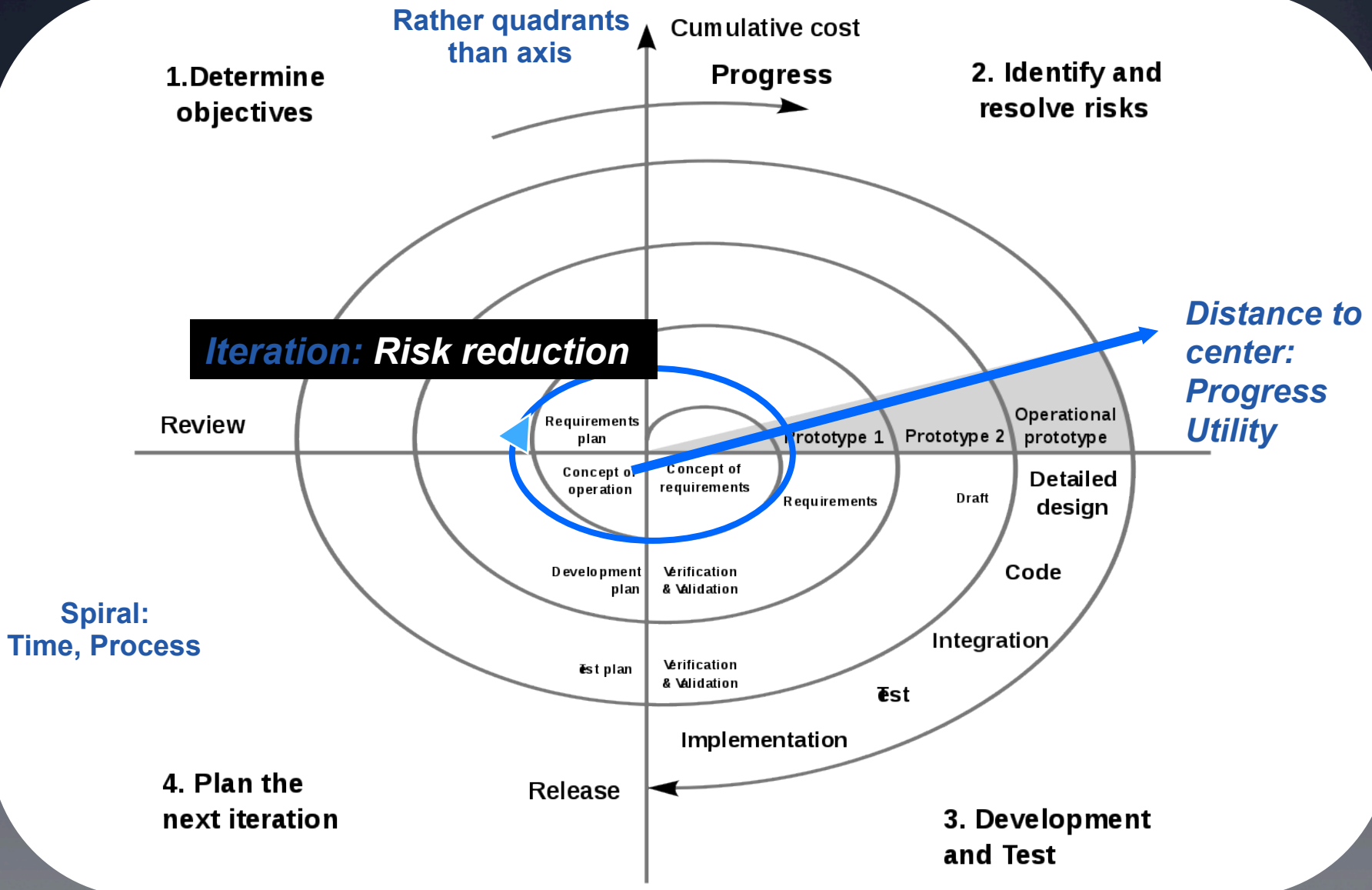
Incremental delivery



Incremental delivery

- Customer value can be delivered with each increment so system functionality is early available for customer's feedback
- Early increments act as prototype to help elicit requirements for later increments
- Reduced risk of project failure
- The highest priority system services tend to receive the most testing

Spiral Model [Boehm]



Spiral development

- Objective setting
  - Specific objectives for the phase are identified
- Risk assessment and reduction
  - Risks are assessed and activities put in place to reduce the key risks
- Development and validation
  - A development model for the system is chosen which can be any of the generic models
- Planning
  - The project is reviewed and the next phase of the spiral is planned

- Is the Spiral model agile?
- What kind of prototypes will be developed in the Spiral model?
- Is consecutive prototyping agile?

What is a prototype?

Why was it constructed?

Horizontal
Vertical

Explorative
Experimental
Evolutionary

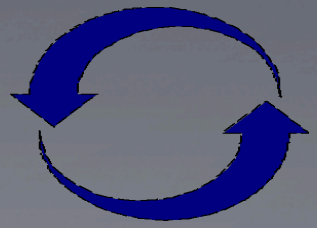
Presentation- PT
Actual PT
Laboratory sample
Pilot system

How do end-users/colleagues/you like the PT?
What was learnt or experienced during construction / presentation of prototype?

+/-

What is the contribution of the prototype?
What is (only) infrastructure?

Technical knowledge during the process that can be valuable somewhere else

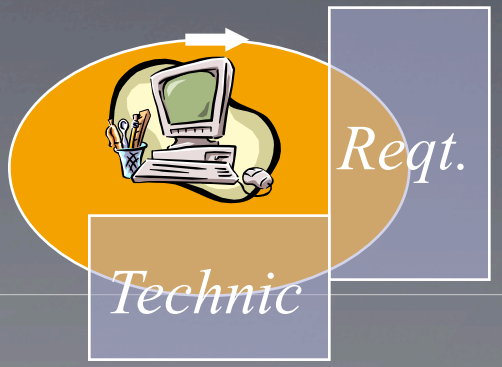


Prototyping approaches

- Explorative
  - Goal: Identify requirements
  - Show alternatives, capture feedback

- Experimental
  - Goal: Assess technical alternatives
  - Test feasibility and implementation

- Evolutionary
  - Goal: Fitting system despite changing requirements
  - Adjust running system constantly to changing requiremetns and constraints



Evolutionary Development

- Applicability
  - Small to midsize projects
  - Parts of larger systems (e.g. user interface)
  - Short lifetime projects

- Restrictions
  - Process visibility for the customer (what is finally delivered?)
  - Risk that systems are only poorly structured
  - Special skills in rapid prototyping languages are necessary

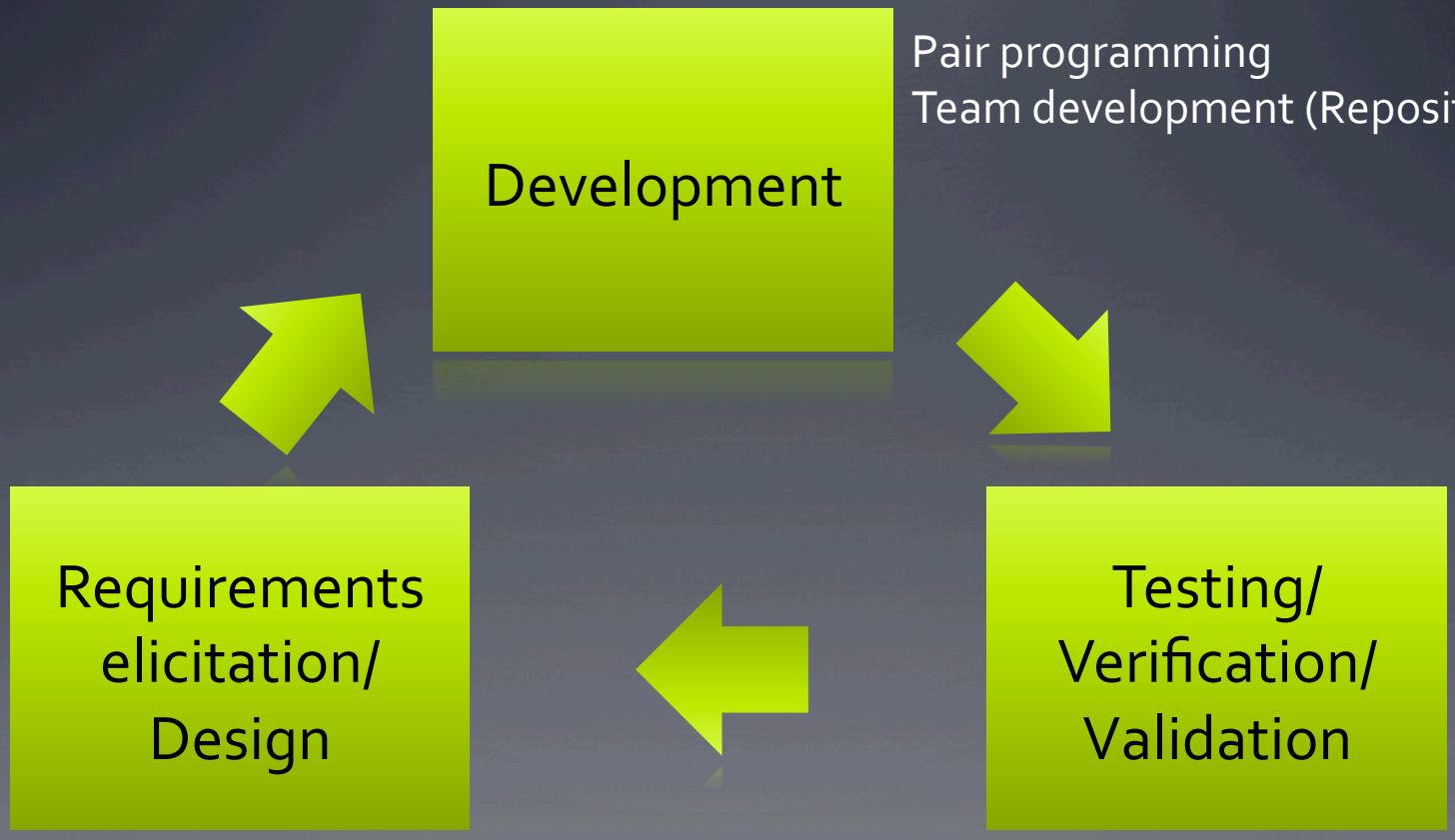
- What is your position about the following statements:
  - "When a customer accepts our prototype, we polish it a bit more and deliver it as product."
  - "We write our prototypes in a different programming language to discourage copy&paste to the main product."

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

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

Agile Software development



Pair programming
Team development (Repositories)

User stories (tasks)
The code is the documentation
Regular meetings with customer

Automated Unit testing
Test-driven development (TDD)
Acceptance tests

Agile processes

- eXtreme programming (XP)
- Scrum
- Kanban
- “Lean”
- Crystal
- ETC

Conclusions

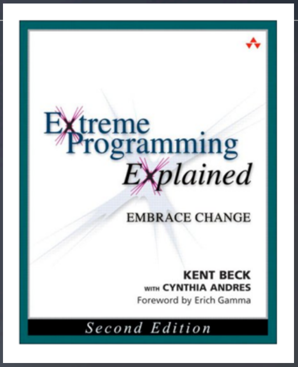
- Efficient software development requires a development process
- Traditional processes
  - Not efficient when requirements change
- Agile software development
  - Processes for efficient software development in the context of change

Agile Methods: eXtreme Programming (XP)

- 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
  - …

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




Best Practices in eXtreme Programming Course Design

Kai Stapel, FG Software Engineering, Leibniz Universität Hannover, Welfengarten 1, 30167 Hannover, Germany, +49 - (0)511 - 762 7463, kai.stapel@inf.uni-hannover.de

Daniel Lübke, FG Software Engineering, Leibniz Universität Hannover, Welfengarten 1, 30167 Hannover, Germany, +49 - (0)511 - 762 19672, daniel.luebke@inf.uni-hannover.de

Eric Knauss, FG Software Engineering, Leibniz Universität Hannover, Welfengarten 1, 30167 Hannover, Germany, +49 - (0)511 - 762 19680, eric.knauss@inf.uni-hannover.de

ABSTRACT

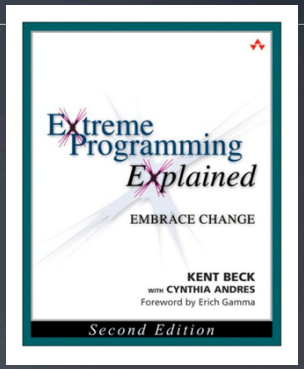
2. RELATED WORK

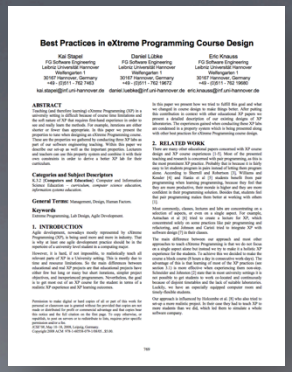
"Students learn how to divide requirements into User Stories and how to prioritize and estimate the costs of these stories. While such tasks seem to be easy in theory, dealing with dependencies in the planning game is normally a challenge for inexperienced developers like students."

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

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)





Best Practices in eXtreme Programming Course Design

Kai Stapel
FG Software Engineering
Leibniz Universität Hannover
Welfengarten 1
30167 Hannover, Germany
+49 - (0)511 - 762 7483
kai.stapel@inf.uni-hannover.de

Daniel Lübke
FG Software Engineering
Leibniz Universität Hannover
Welfengarten 1
30167 Hannover, Germany
+49 - (0)511 - 762 19872
daniel.luebke@inf.uni-hannover.de

Eric Knauss
FG Software Engineering
Leibniz Universität Hannover
Welfengarten 1
30167 Hannover, Germany
+49 - (0)511 - 762 19880
eric.knauss@inf.uni-hannover.de

ABSTRACT

Keywords

General Terms

1. INTRODUCTION

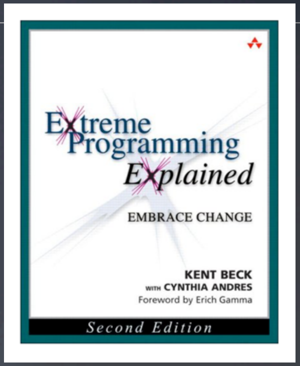
2. RELATED WORK

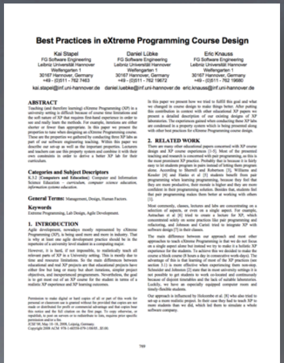
"Students learn the benefits of small releases that already offer value to the customer and how to technically put a system into production including packaging."

- (+) 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





Best Practices in eXtreme Programming Course Design

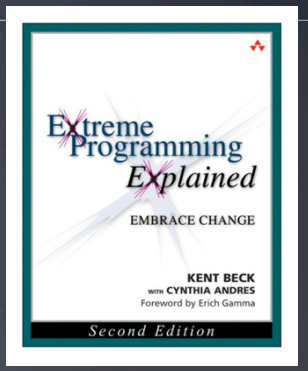
Kai Stapel, Daniel Lübke, Eric Knauss

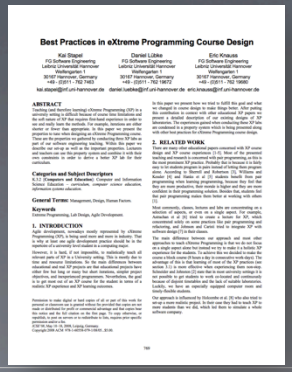
“Students learn how to develop a metaphor that helps every team member to better understand how the whole system works.”

- (+) Technical support, 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





Best Practices in eXtreme Programming Course Design

Kai Stapel
FG Software Engineering
Leibniz Universität Hannover
Welfengarten 1
30167 Hannover, Germany
+49 - (0)511 - 762 7463
kai.stapel@inf.uni-hannover.de

Daniel Lübke
FG Software Engineering
Leibniz Universität Hannover
Welfengarten 1
30167 Hannover, Germany
+49 - (0)511 - 762 19672
daniel.luebke@inf.uni-hannover.de

Eric Knauss
FG Software Engineering
Leibniz Universität Hannover
Welfengarten 1
30167 Hannover, Germany
+49 - (0)511 - 762 19680
eric.knauss@inf.uni-hannover.de

ABSTRACT

Categories and Subject Descriptors

General Terms

Keywords

1. INTRODUCTION

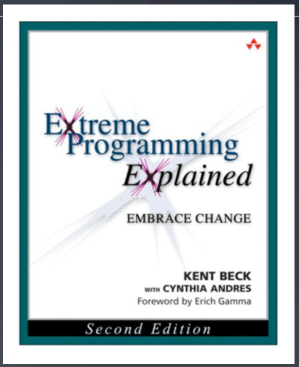
2. RELATED WORK

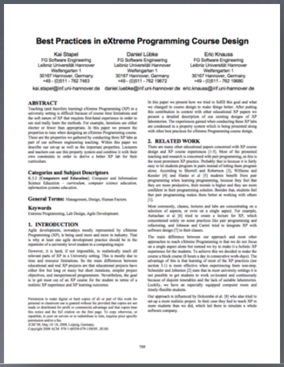
"Students learn the benefits of simple software design which improves their ability to change the system quickly and accommodate it to changing requirements."

- (+) More iterations, technical support

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"



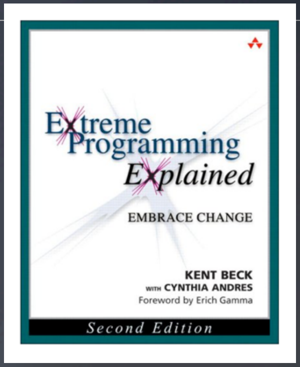


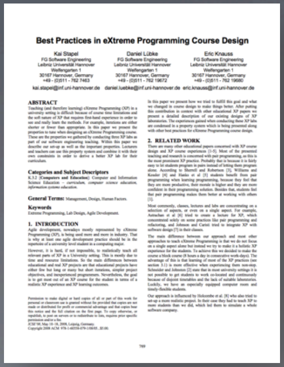
"Students learn how to use unit test frameworks and the test-first approach to build high quality software and to recognize the advantages of well-tested code when making changes."

- (+) 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!!!




Best Practices in eXtreme Programming Course Design

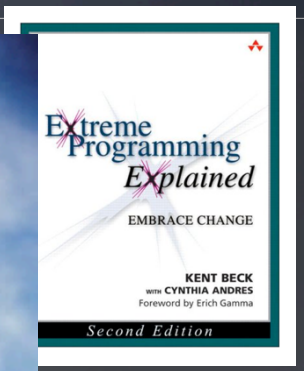
Kai Stapel
Daniel Lübke
Eric Knauss

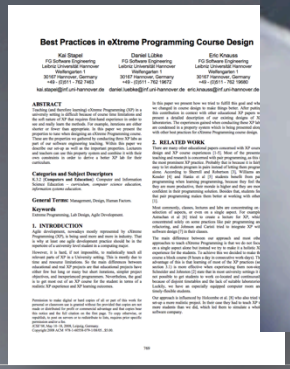
"Students learn to refactor the software to remove duplication, improve communication and simplify the code base. Especially refactoring large systems can be troublesome and is a worthy experience that can only be made in long lasting projects."

- (+) More iterations, customer interest, technical support

Pair Programming

- Drive
- Navig
  - Wil
    Car




Best Practices in eXtreme Programming Course Design

Kai Stapel
FG Software Engineering
Leibniz Universität Hannover
Welfengarten 1
30167 Hannover, Germany
+49 - (0)511 - 762 7483
kai.stapel@inf.uni-hannover.de

Daniel Lübke
FG Software Engineering
Leibniz Universität Hannover
Welfengarten 1
30167 Hannover, Germany
+49 - (0)511 - 762 19872
daniel.luebke@inf.uni-hannover.de

Eric Knauss
FG Software Engineering
Leibniz Universität Hannover
Welfengarten 1
30167 Hannover, Germany
+49 - (0)511 - 762 19890
eric.knauss@inf.uni-hannover.de

ABSTRACT

Categories and Subject Descriptors

General Terms

Keywords

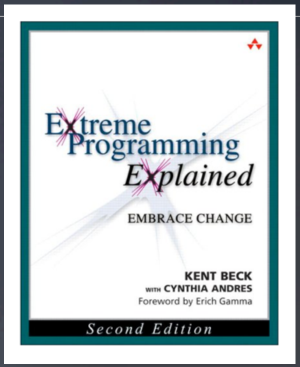
1. INTRODUCTION

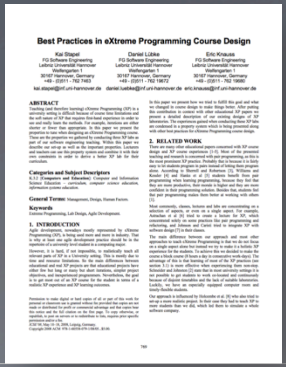
2. RELATED WORK

n a partner

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




Best Practices in eXtreme Programming Course Design

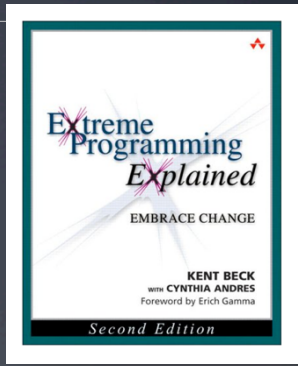
Kai Stapel · Daniel Lübke · Eric Knauss

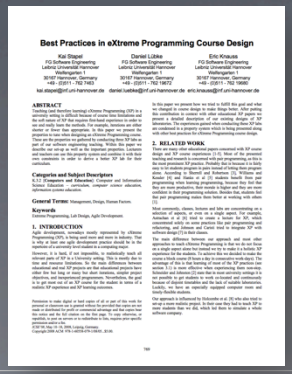
"Students get to know the advantages of collective code ownership and the challenges that arise with parallel updates and changes to their own code by other team members. "

- (+) Block course, small team size

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





Best Practices in eXtreme Programming Course Design

Kai Stapel, Daniel Lübke, Eric Knauss
FG Software Engineering, Leibniz Universität Hannover, Wellingerstr. 1, 30167 Hannover, Germany
+49 - (0)511 - 762 7462 / 19672 / 19660
kai.stapel@inf.uni-hannover.de daniel.luebke@inf.uni-hannover.de eric.knauss@inf.uni-hannover.de

ABSTRACT

Keywords

1. INTRODUCTION

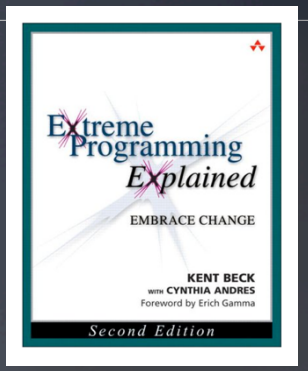
2. RELATED WORK

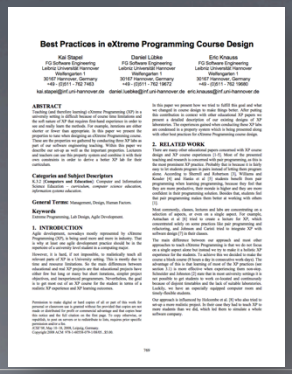
"To counter conflicting updates to the code base, students learn to integrate and build the software frequently. "

- (+) Block course, longer iterations, more iterations, small team size
- (discovered later: technical feedback)

Sustainable pace (aka 40h week)

- Be fresh every morning, tired every night
- One week of overtime must not be followed by another one





Best Practices in eXtreme Programming Course Design

Kai Stapel
FG Software Engineering
Leibniz Universität Hannover
Welfengarten 1
30167 Hannover, Germany
+49 - (0)511 - 762 7463
kai.stapel@inf.uni-hannover.de

Daniel Lübke
FG Software Engineering
Leibniz Universität Hannover
Welfengarten 1
30167 Hannover, Germany
+49 - (0)511 - 762 19672
daniel.luebke@inf.uni-hannover.de

Eric Knauss
FG Software Engineering
Leibniz Universität Hannover
Welfengarten 1
30167 Hannover, Germany
+49 - (0)511 - 762 19680
eric.knauss@inf.uni-hannover.de

ABSTRACT

Categories and Subject Descriptors

General Terms

Keywords

1. INTRODUCTION

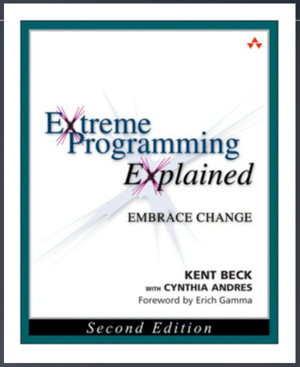
2. RELATED WORK

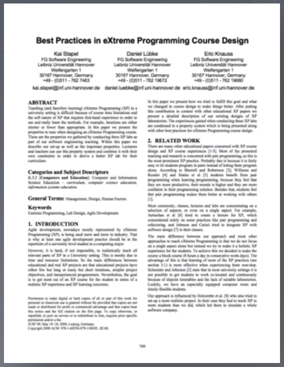
"In contrast to normal life in university, students experience to work continuously for 40-hours per week in a designated team room."

- (+) **Block course**

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*



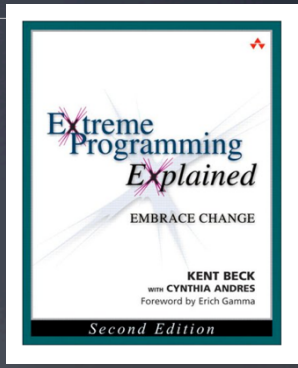

Best Practices in eXtreme Programming Course Design
Kai Stapel
Daniel Lübke
Eric Knauss

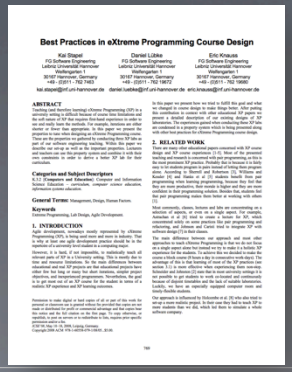
"Students have to interact with a designated On-Site Customer who is available full-time to answer questions."

- (+) Block course, customer interest

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





Best Practices in eXtreme Programming Course Design

Kai Stapel
FG Software Engineering
Leibniz Universität Hannover
Welfengarten 1
30167 Hannover, Germany
+49- (0)511 - 762 7463
kai.stapel@inf.uni-hannover.de

Daniel Lübke
FG Software Engineering
Leibniz Universität Hannover
Welfengarten 1
30167 Hannover, Germany
+49- (0)511 - 762 19672
daniel.luebke@inf.uni-hannover.de

Eric Knauss
FG Software Engineering
Leibniz Universität Hannover
Welfengarten 1
30167 Hannover, Germany
+49- (0)511 - 762 19680
eric.knauss@inf.uni-hannover.de

ABSTRACT

Categories and Subject Descriptors

General Terms

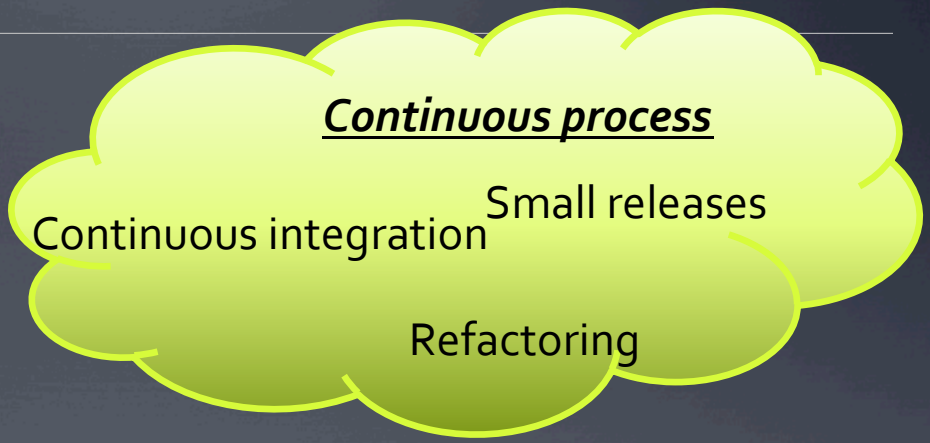
Keywords

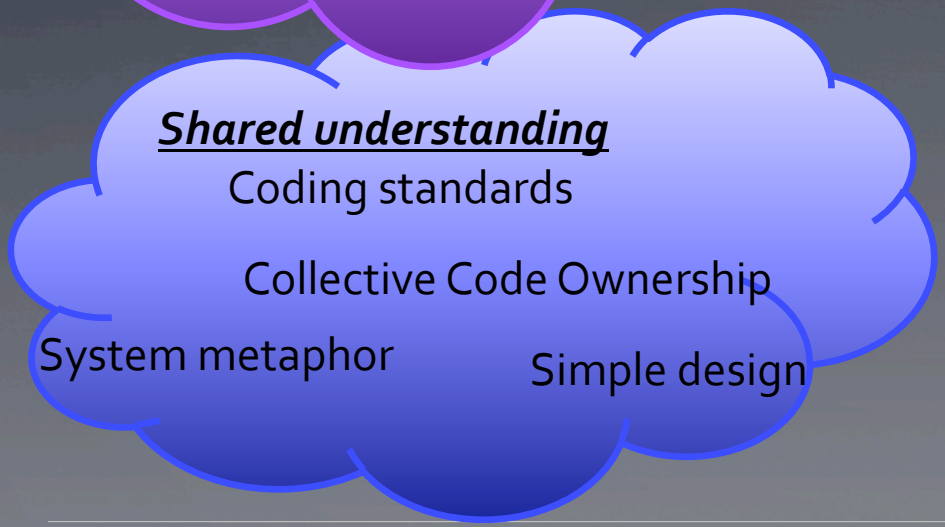
1. INTRODUCTION

2. RELATED WORK

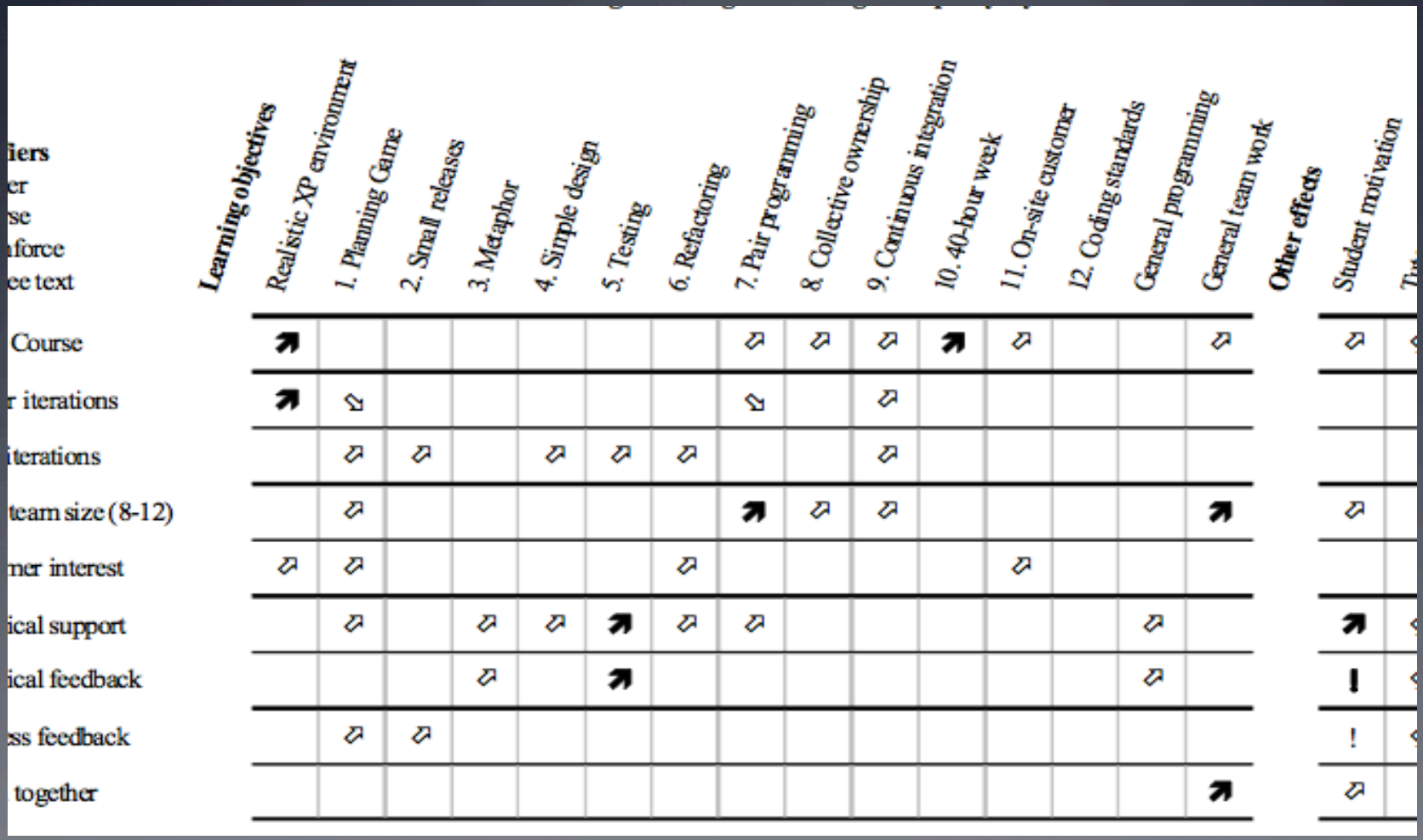
"Students experience the importance of uniform coding conventions throughout the team especially when combined with collective code ownership. "

XP Practices



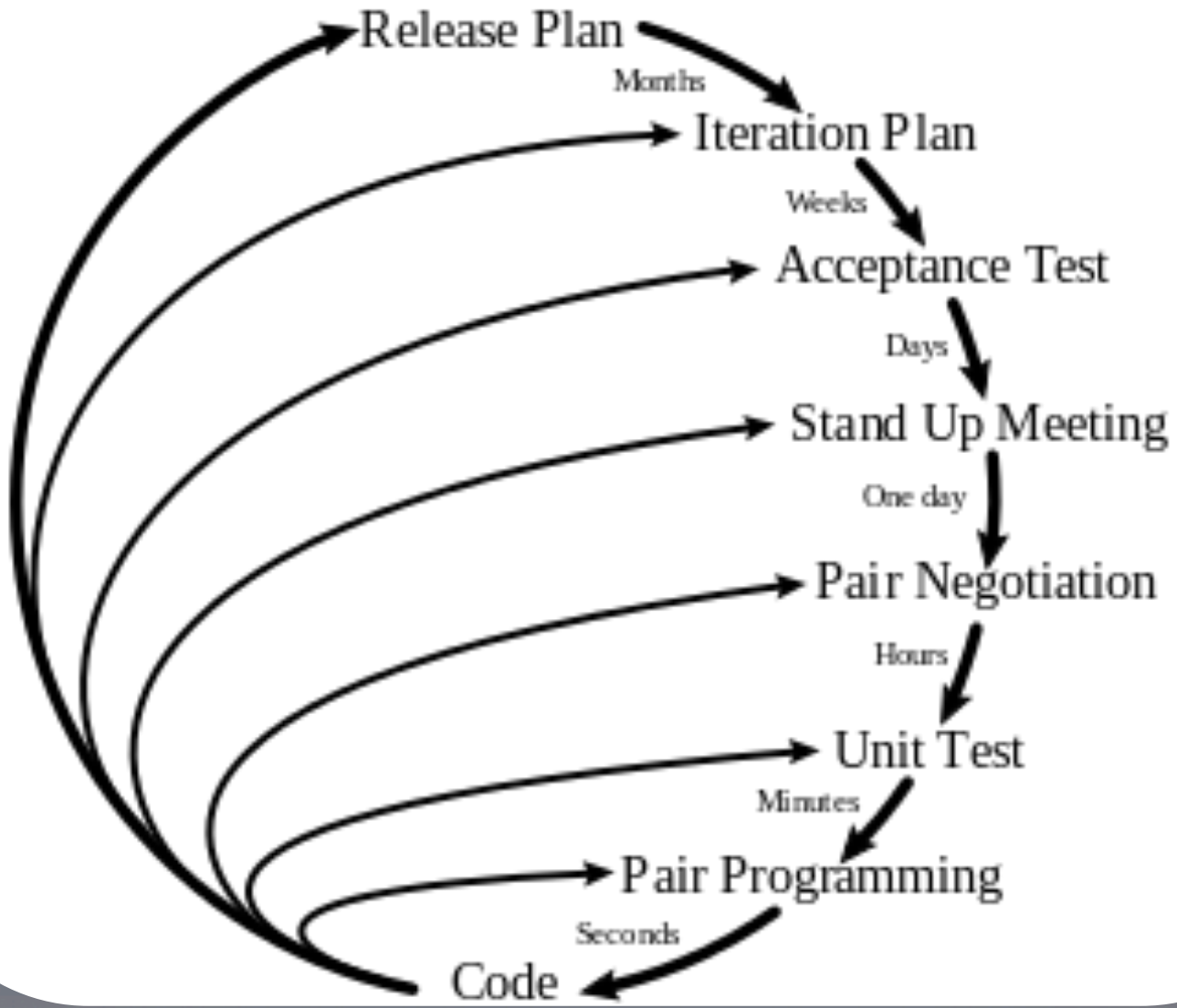


Overview: XP in Teaching



iers / er / se / force / ce text	Learning objectives	Realistic XP environment	1. Planning Game	2. Small releases	3. Metaphor	4. Simple design	5. Testing	6. Refactoring	7. Pair programming	8. Collective ownership	9. Continuous integration	10. 40-hour week	11. On-site customer	12. Coding standards	General programming	General team work	Other effects	Student motivation
Course	↗							↘	↘	↘	↗	↘			↘		↘	
r iterations	↗	↘						↘		↘								
iterations		↘	↘		↘	↘	↘			↘								
team size (8-12)		↘						↗	↘	↘					↗		↘	
ner interest	↘	↘					↘					↘						
ical support		↘		↘	↘	↗	↘	↘						↘			↗	
ical feedback				↘		↗								↘			!	
ss feedback		↘	↘														!	
together															↗		↘	

XP: Planning/Feedback Loops



Thursday

- Until Thursday: Reflect about the XP practices and how they fit together!
- Continue with agile principles, processes and practices
- Emil.Alegroth@chalmers.se