Agile Software Development (DIT191 / EDA397)

Eric Knauss
<eric.knauss@cse.gu.se>

Hi, nice meeting you! (Short CV)

- Born in Hattingen, Germany
- University in Dortmund and Hanover
- PhD in Hanover: **Improving Requirements Documentation based on Heuristics and Experience**
- Postdoc in Hanover: **Global Software Development**
- Postdoc in Victoria BC: **RE in Distributed Large-Scale Software Projects**

- **Topics**
  - **Requirements Engineering**
  - **Agile Methods**
  - **RE and Project Management**
  - **Experience and Knowledge Management**

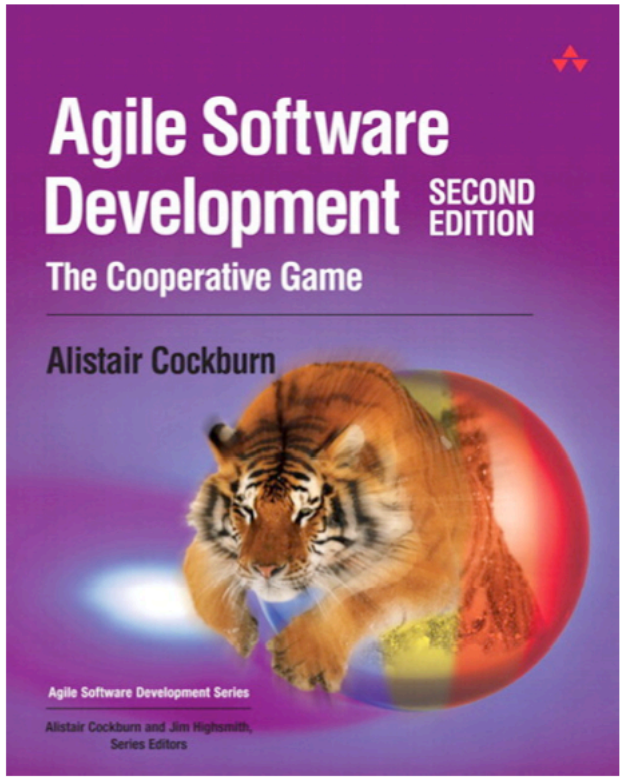
Course setup

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

- Course representants

- In parallel:
  - PhD course

- Course Material
  - http://oerich.github.io/EDA397/
  - Cockburn,A., (2009) Agile Software Development, 2ed
  - 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
  - Report (Individual)
    - Experience / Post-Mortem report
  - Written exam

Examination

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

Project

- Develop an Android app 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

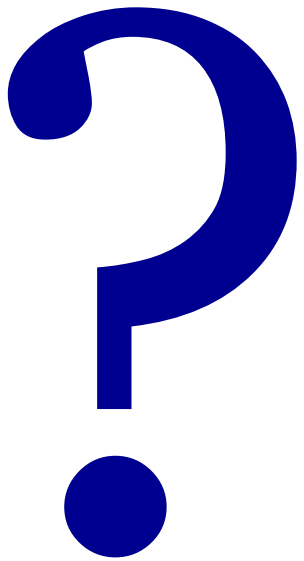
Sprint 1: Getting started

What is agility in Software Development?

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

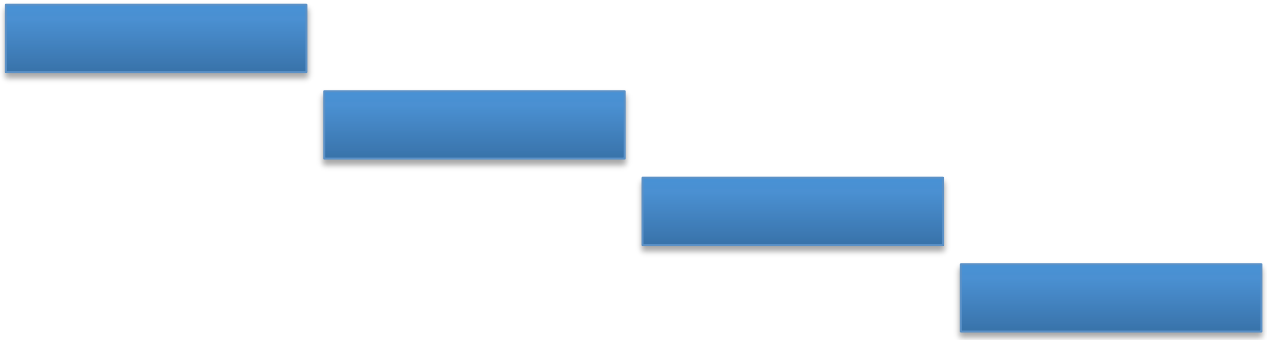


- Do you know examples of
  - Systematic, disciplined, quantifiable approaches to the development, operation, and maintenance of software?
  - Applying engineering to software?

Where are we coming from?

Systematic sequential development

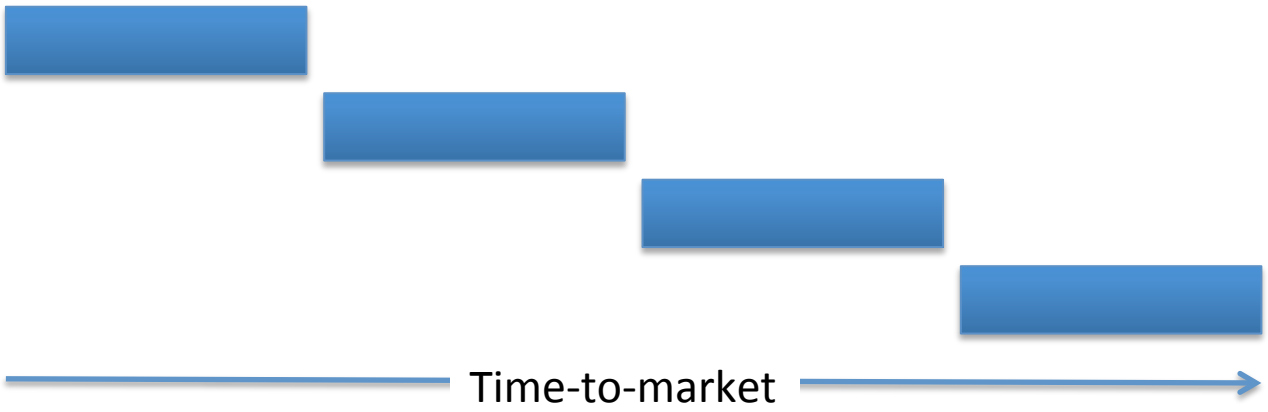
- Requirements
- Design
- Programming
- Test



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

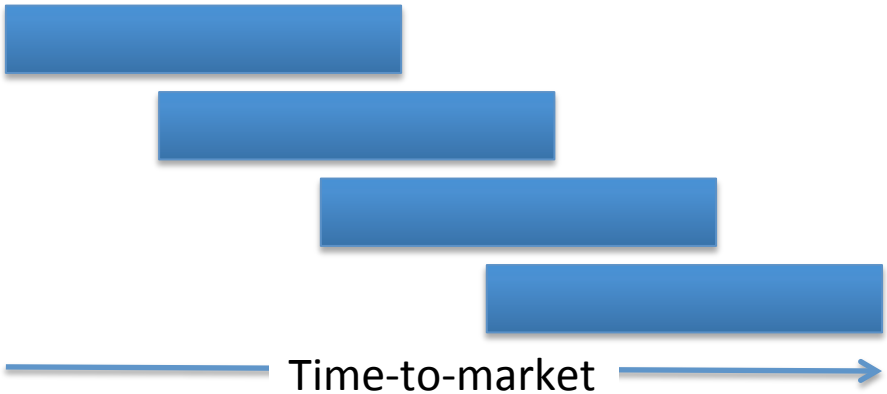
Towards concurrent development

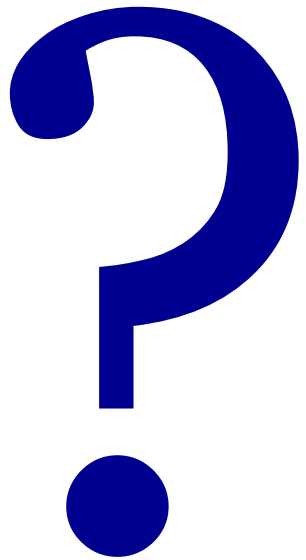
- Requirements
- Design
- Programming
- Test



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

- Requirements
- Design
- Programming
- Test



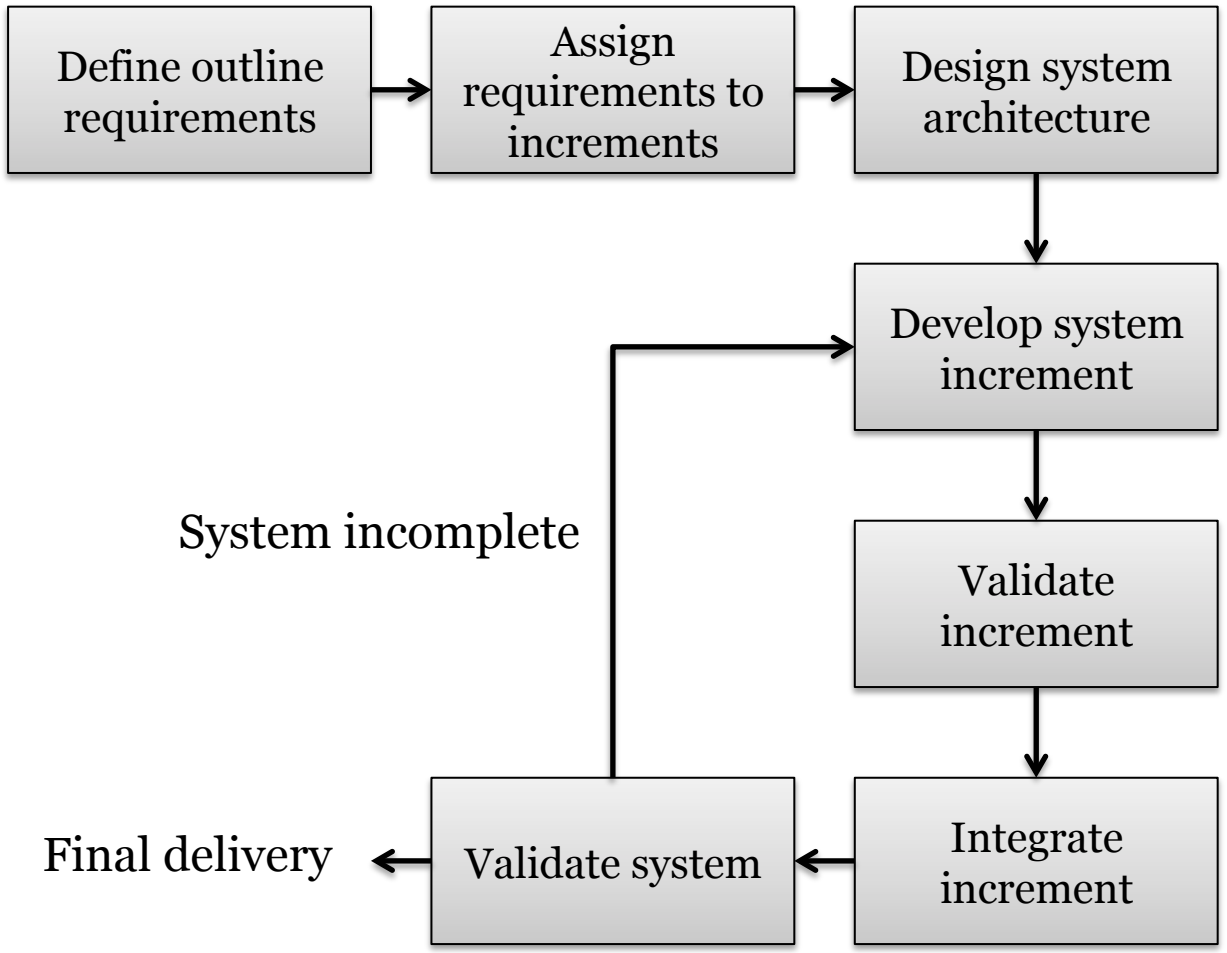


- What is the consequence of concurrent development?
  - (hint: why are concurrent tasks depicted longer than sequential tasks?)
- What has this to do with agile?
- What do you know about agile?

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



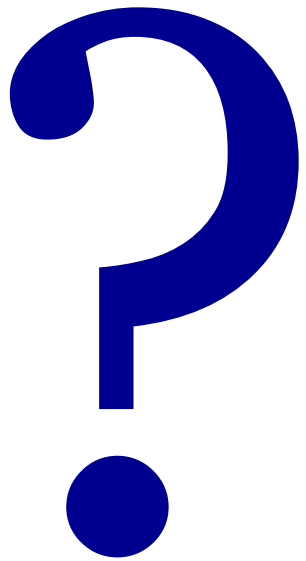
Incremental delivery

- 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

Agile Manifesto

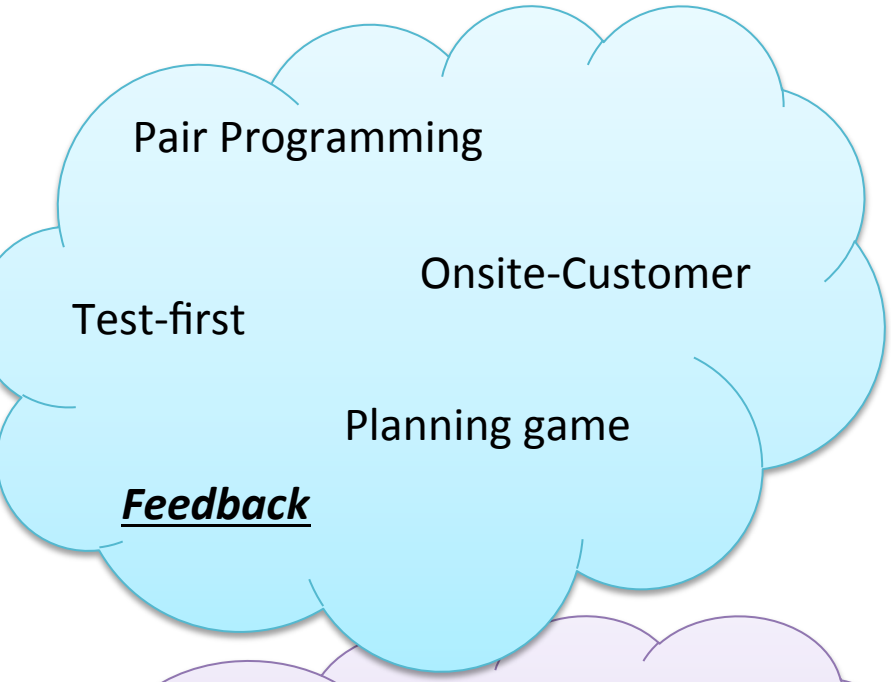
http://agilemanifesto.org

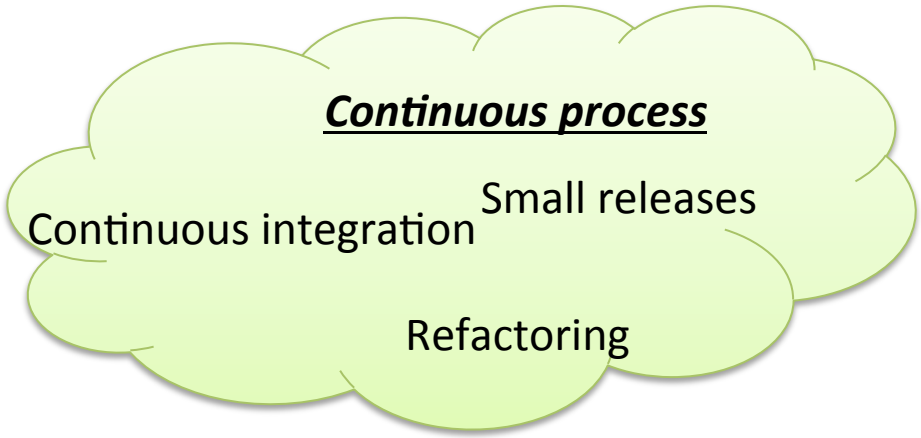
- Began as a provocation: Plan-driven development did not safe the Software world…
- Now a very serious movement, well adapted in industry.
- There are a couple of established agile methods: How to integrate these values in everyday software development

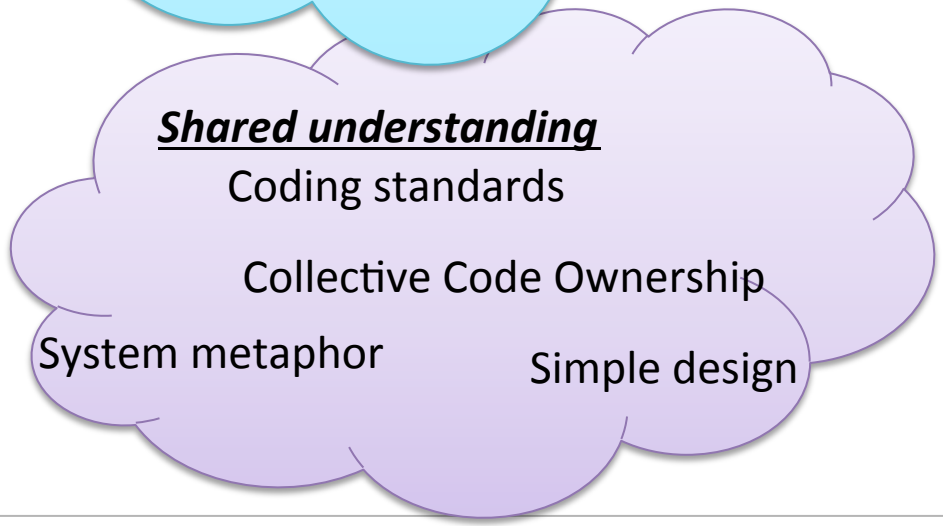


- Can the following projects be agile?
  - App development
  - Online shop
  - Mission controller for Airplane
  - Controller for nuclear plant

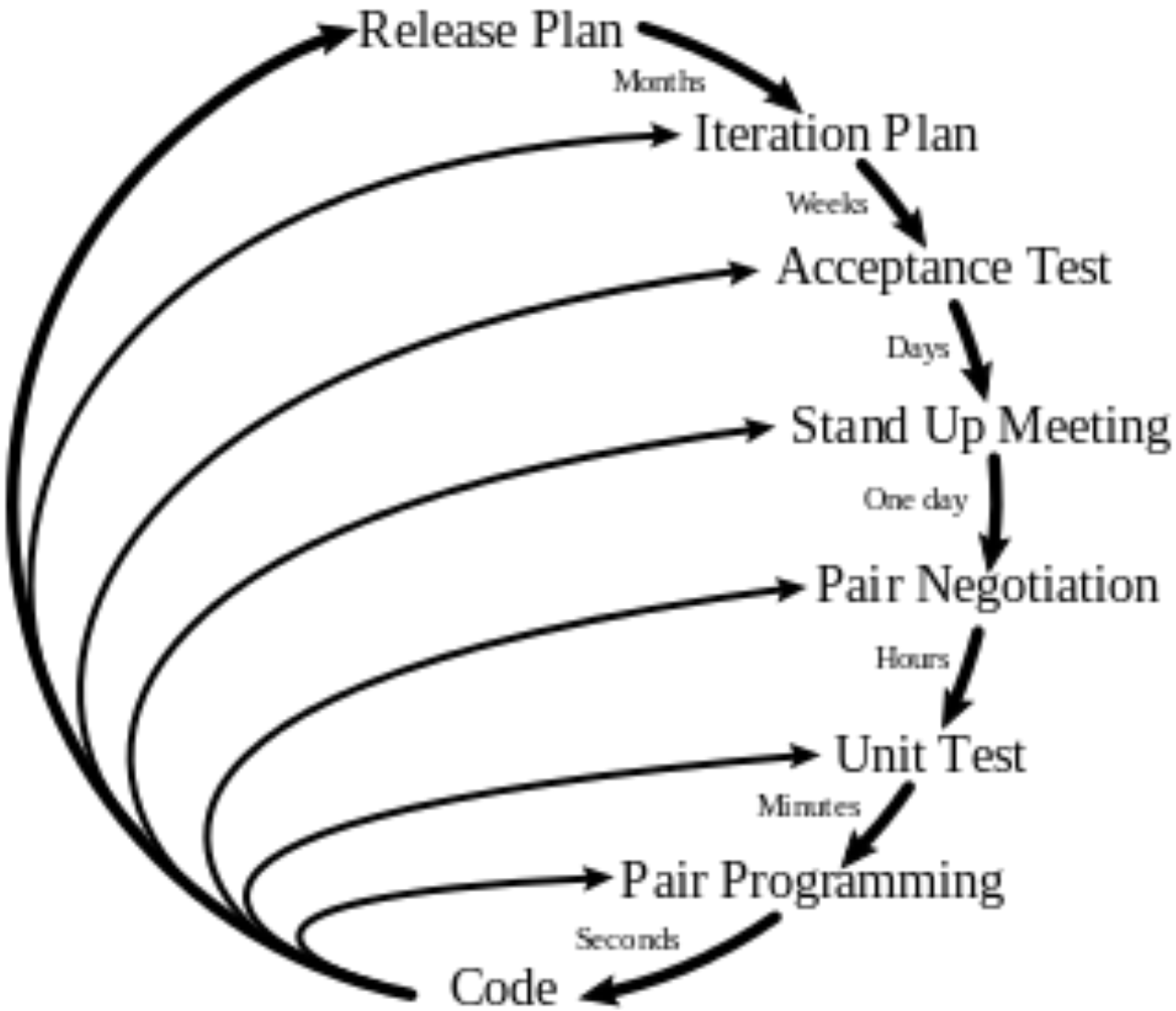
XP Practices







XP: Planning/Feedback Loops



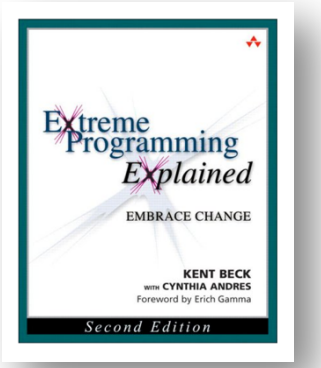
http://en.wikipedia.org/wiki/File:Extreme_Programming.svg

Agile Methods: eXtreme Programming (XP)

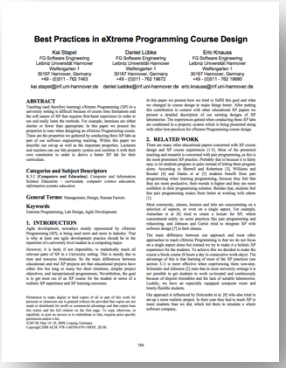
- Extreme programming:
  - An approach based on the development and delivery of very small increments of functionality
  - No fine grained process description, but 12 practices arranged around short development circles (4-6 weeks)
  - "Turn-to-ten" metaphor (refers to volume setting of older amplifiers):
    - Reviewing is good? → Review continuously: Pair Programming
    - Early Tests are good? → Write tests before code: Test-First
    - Customer interaction is good? → Have Onsite-Customer
    - …

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]


Best Practices in eXtreme Programming Course Design

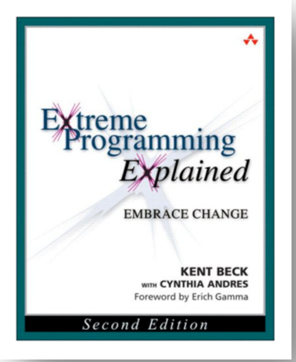
[SLK2008]

"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

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

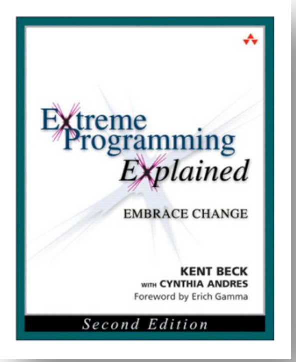
Kai Stapel
Daniel Lübke
Eric Knauss

"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

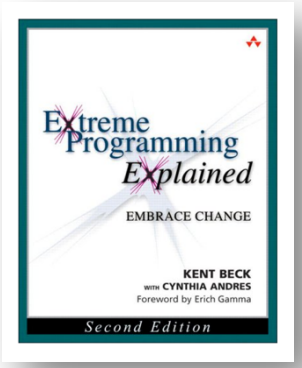
Kai Stapel — Daniel Lübke — Eric Knauss

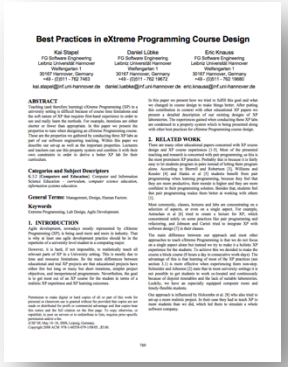
“Students learn how to develop a metaphor that helps every team member to better understand how the whole system works.”

- (+) 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, Daniel Lübke, Eric Knauss
FG Software Engineering, Leibniz Universität Hannover, Welfengarten 1, 30167 Hannover, Germany

ABSTRACT

Categories and Subject Descriptors

General Terms

Keywords

1. INTRODUCTION

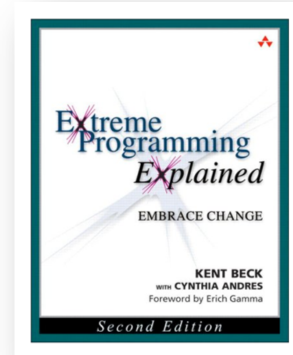
2. RELATED WORK

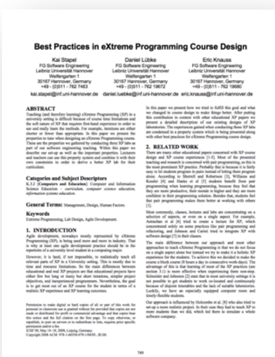
"Students learn the benefits of simple software design which improves their ability to change the system quickly and accommodate it to changing requirements."

- (+) 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"




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

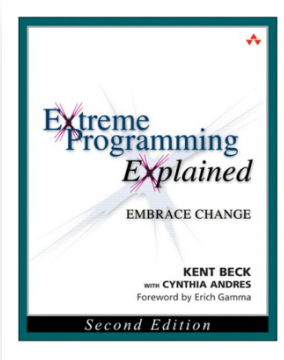
Eric Knauss
FG Software Engineering
Leibniz Universität Hannover
Welfengarten 1
30167 Hannover, Germany
+49 - (0)511 - 762 19680
eric.knauss@inf.uni-hannover.de

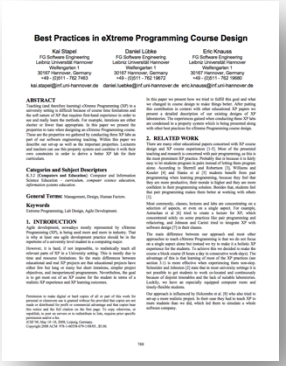
"Students learn how to use unit test frameworks and the test-first approach to build high quality software and to recognize the advantages of well-tested code when making changes."

- (+) 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
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 19682
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

1. INTRODUCTION

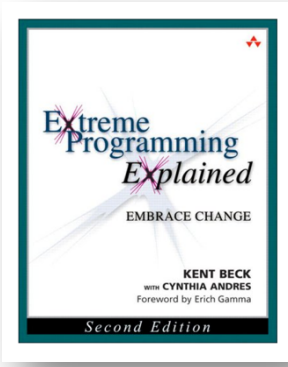
2. RELATED WORK

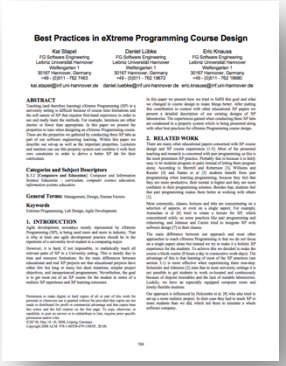
"Students learn to refactor the software to remove duplication, improve communication and simplify the code base. Especially refactoring large systems can be troublesome and is a worthy experience that can only be made in long lasting projects."

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

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
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 19682
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

1. INTRODUCTION

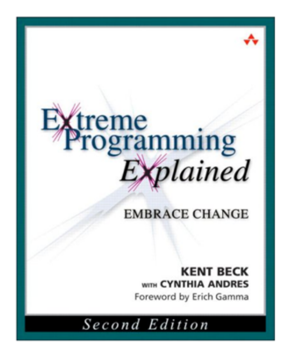
2. RELATED WORK

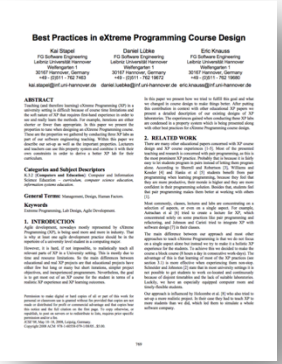
"Students learn to refactor the software to remove duplication, improve communication and simplify the code base. Especially refactoring large systems can be troublesome and is a worthy experience that can only be made in long lasting projects."

- (+) 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?





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

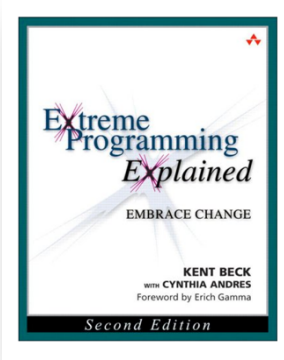
2. RELATED WORK

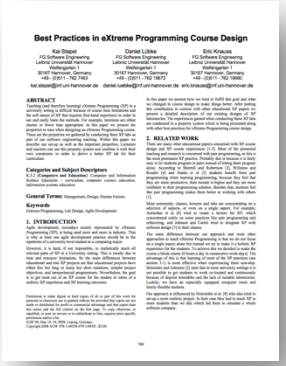
“Students learn and experience the principles of pair programming, the advantages of writing software with a partner and the social challenges that are associated.”

- (+) 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





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 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

Categories and Subject Descriptors

General Terms

Keywords

1. INTRODUCTION

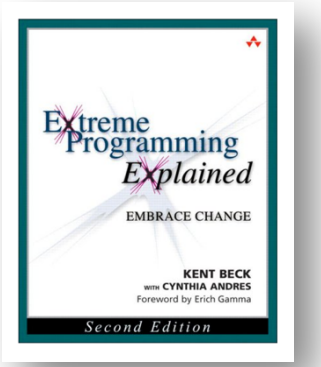
2. RELATED WORK

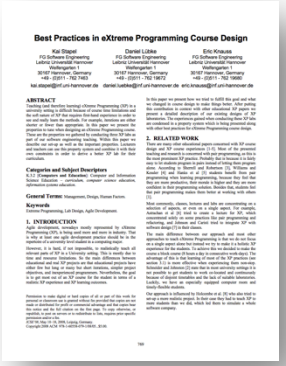
"Students get to know the advantages of collective code ownership and the challenges that arise with parallel updates and changes to their own code by other team members. "

- (+) 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, Welfengarten 1, 30167 Hannover, Germany

ABSTRACT

Categories and Subject Descriptors

General Terms

Keywords

1. INTRODUCTION

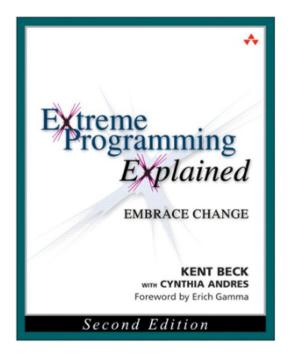
2. RELATED WORK

"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

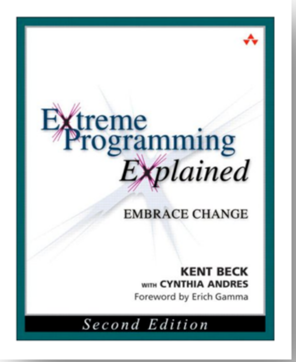


"In contrast to normal life in university, students experience to work continuously for 40-hours per week in a designated team room."

- (+) **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

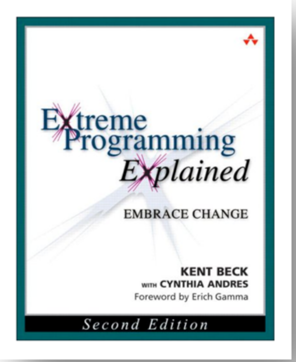
Kai Stapel, Daniel Lübke, Eric Knauss

“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

2. RELATED WORK

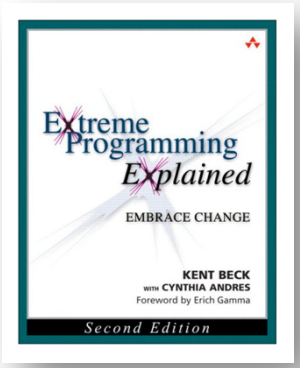
"Students experience the importance of uniform coding conventions throughout the team especially when combined with collective code ownership. "

Agile Principles and Practices

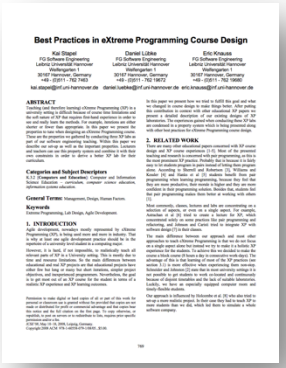
- Goal: Try them out in your project!

	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*

References



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


Best Practices in eXtreme Programming Course Design

Kai Stapel
FG Software Engineering
Leibniz Universität Hannover
Welfengarten 1
30167 Hannover, Germany
+49 - (0)511 - 762 7453
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

Categories and Subject Descriptors

General Terms

Keywords

1. INTRODUCTION

2. RELATED WORK

[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**