Vorlesung "Software-Engineering"

Prof. Ralf Möller, TUHH, Arbeitsbereich STS

- Vorige Vorlesung
  - Modellbasiertes Software-Engineering
  - Model-Driven Architecture
  - Software-Engineering-Werkzeuge: Ant
- Heute
  - Probleme klassischer Vorgehensmodelle
  - Extreme Programming (XP)
  - Feature-Driven-Development
  - Generalisierung: Agile Software-Entwicklungsmethoden

Probleme klassischer Vorgehensmodelle (1)

- Entwicklersicht
  - Bürokratie (Dokumentation wichtiger als Ergebnis)
  - Starres Rollenschema
  - Blick auf das Ganze fehlt
    - Fließbandsicht
    - Motivation fehlt
  - Kundenkontakt fehlt
    - keine Nachfragemöglichkeit
    - Fehlentwicklung durch Unkenntnis der Anwendung
- Kundensicht (später)

eXtreme Programming:
Ein Vorschlag zur Lösung der Probleme klassicher Vorgehensmodelle

Präsentationen von

D. Dranidis

September 2000

CITY College

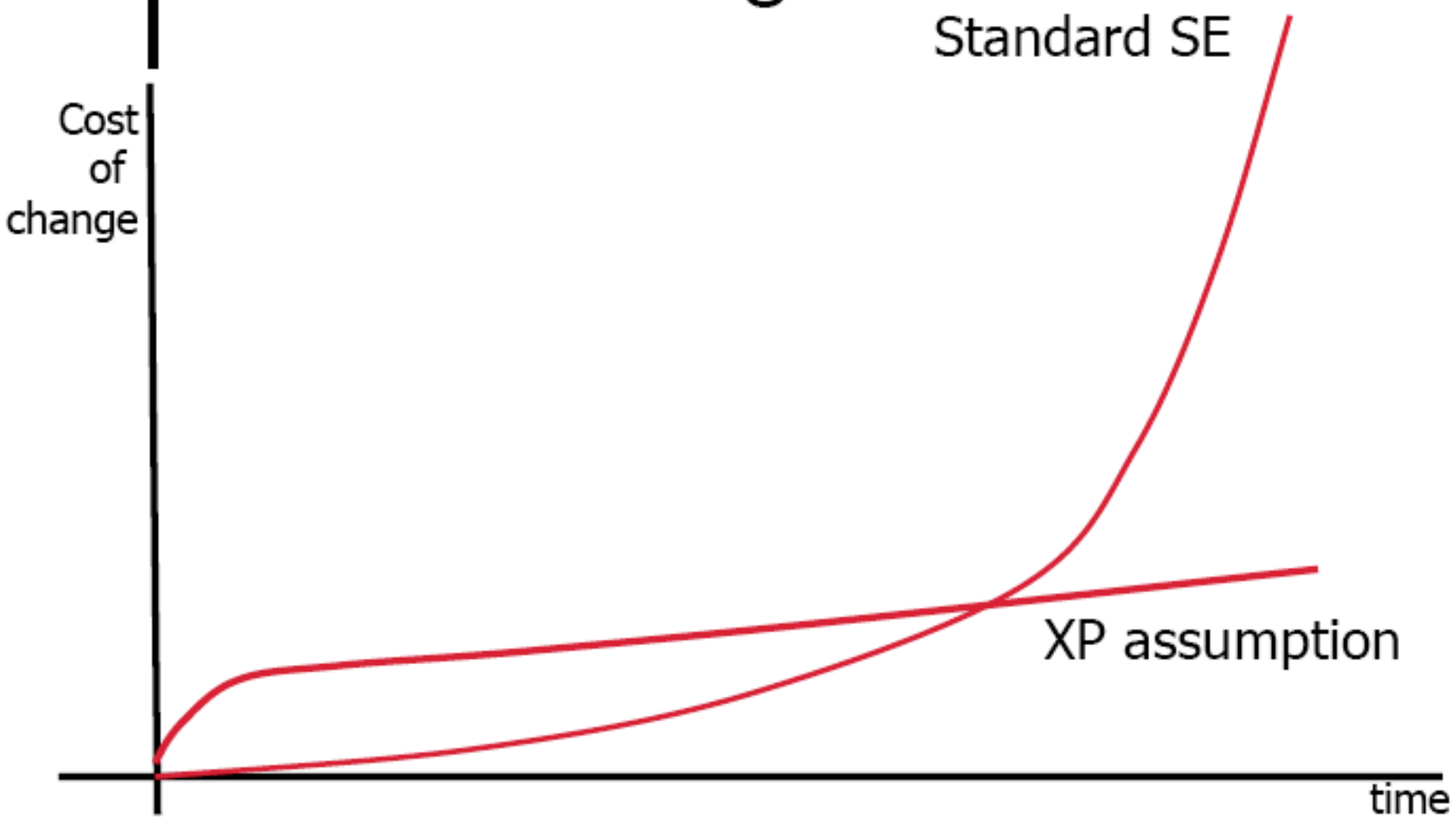
What is XP?

- Who is behind XP?
  - Kent Beck, Ward Cunningham, Ron Jeffries
- How old is XP?
  - almost 4 years old
- Short definition
  - lightweight process model for OO software development
- What's in the name?
  - code is in the centre of the process
  - practices are applied extremely
- What is new in XP?
  - none of the ideas or practices in XP are new
  - the combination of practices and their extreme application is new

Practices

- XP is based on the extreme application of 12 practices (guidelines or rules) that support each other:
  - Planning game
  - Frequent releases
  - System metaphor
  - Simple design
  - Tests
  - Refactoring
  - Pair programming
  - Collective code ownership
  - Continuous Integration
  - Forty-hour week
  - On-site customer
  - Coding standards

Cost of change



Planning Game

- **Pieces:** user stories
- **Players:** customer & developer
- **Moves:**
  - **User story writing**
    - requirements are written **by the customer** on small index cards
    - user stories are written in business language
    - and describe things that the system needs to do
    - each user story is assigned a **business value**
    - Example (payroll system):
      - *An employee making $10 an hour works four hours of overtime on Friday and two on Sunday. She should receive $60 for the Friday and $40 for the Sunday*
    - for a few months projects there may be 50-100 user stories

Planning Game (2)

- **Moves:**
  - **Story estimation**
    - each user story is assigned a cost **by the developer**
    - cost is measured on ideal weeks (1-3 weeks)
    - a story is split **by the customer** if it takes longer than 3 weeks to implement
  - **Commitment**
    - customer and developer decide which user stories constitute the next release
  - **Value and Risk first**
    - developer orders the user stories of the next release so that
      - more valuable or riskier stories are moved earlier in the schedule
      - a fully working (sketchy) system is completed (in a couple of weeks)

Frequent Releases

- The development process is highly iterative
- A release cycle is usually up to 3 months
- A release cycle consists of iterations up to 3 weeks
- In each iteration the selected user stories are implemented
- Each user story is split in programming tasks of 1-3 days

- small and frequent releases provide frequent feedback from the customer

System Metaphor

- Synonym for system-architecture ?
- The system metaphor provides a broad view of the project's goal
- It defines the overall theme to which developers and clients can relate
- Common concept of what the system is like
- The system is built around one (or more) system metaphor from which classes, methods, variables and basic responsibilities are derived
- "Chrysler is a manufacturing company; we make cars. Using a manufacturing metaphor to define the project was an important first step in getting the team (and management) on a level playing field. The concepts of lines, parts, bins, jobs, and stations are metaphors understood throughout the company. The team had the benefit of a very rich domain model developed by members of the team in the project's first iteration. It gave the members of the project an edge in understanding an extremely complex domain."

Simple Design

- Do the simplest thing that could possibly work
  - create the best (simple) design you can
- You aren't going to need it
  - do not spend time implementing potential future functionality (requirements will change)
- **Put in what you need when you need it**
- Simple design ensures that there is less
  - to communicate
  - to test
  - to refactor

Tests

- Tests play the most important and central role in XP
- Tests are written before the code is developed
  - forces concentration on the interface
  - accelerates development
  - safety net for coding and refactoring
- **All** tests are automated (test suites, testing framework)
- If user stories are considered as the requirements then Tests can be considered as the specification of the system
- 2 kinds of test:
  - Acceptance tests (functional tests)
    - clients provide test cases for their stories
    - developers transform these in automatic tests
  - Unit tests
    - developers write tests for their classes (before implementing the classes)
    - All unit tests must run 100% successfully all the time

Refactoring

- **Change it even if it is not broken!**
- Process of improving code while preserving its function
- The aim of refactoring is to
  - make the design simpler
  - make the code more understandable
  - improve the tolerance of code to change
- The code should not need any comments
  - There is no documentation in XP
  - The code and the user stories are the only documents
- Useful names should be used (system metaphor)
- Refactoring is continuous design
- Remove duplicate code
- Tests guarantee that refactoring didn’t break anything that worked!

Pair programming

- Two programmers sit together in front of a workstation
  - one enters code
  - one reviews the code and thinks
- “Pair programming is a dialog between two people trying to simultaneously program and understand how to program better”, *Kent Beck*
- Second most important practice after tests
- Pairs change continuously (few times in a day)
  - every programmer knows all the aspects of the system
  - a programmer can be easily replaced in the middle of the project
- Costs 10-15% more than stand-alone programming
- Code is simpler (fewer LOC) with less defects (15%)
- Ensures continuous code inspection (SE)

Collective code ownership

- The code does not belong to any programmer but to the team
- Any programmer can (actually **should**) change any of the code at any time in order to
  - make it simpler
  - make it better
- Encourages the entire team to work more closely together
- Everybody tries to produce a high-quality system
  - code gets cleaner
  - system gets better all the time
  - everybody is familiar with most of the system

Continuous integration

- Daily integration at least
- The whole system is built (integrated) every couple of hours
- XP feedback cycle:
  - develop unit test
  - code
  - integrate
  - run all units tests (100%)
  - release
- A working tested system is always available

40 hour week

- “Overtime is defined as time in the office when you don’t want to be there” *Ron Jeffries*
- Programmers should not work more than one week of overtime
- If more is needed then something is wrong with the schedule

- Keep people happy and balanced
- Rested programmers are more likely to refactor effectively, think of valuable tests and handle the strong team interaction

On-site customer

- User stories are not detailed, so there are always questions to ask the customer
- The customer must always be available
  - to resolve ambiguities
  - set priorities
  - provide test cases
- Customer is considered part of the team

Coding standards

- Coding standards make pair progamming and collective code ownership easier
- Common name choosing scheme
- Common code formatting

Listen-Test-Code-Design

- Traditional Software Lifecycle:
  - Listen - Design - Code - Test
- XP lifecycle
  - Listen - Test - Code - Design

- **Listen** to customers while gathering requirements
- Develop **test** cases (functional tests and unit tests)
- **Code** the objects
- **Design** (refactor) as more objects are added to the system

Requirements

- small teams (up to 10-15 programmers)
- common workplace and working hours
- all tests must be automated and executed in short time
- on-site customer
- developer and client must commit 100% to XP practices

XP is successful because...

- XP can handle changing customer requirements, even late in the life cycle
- XP stresses customer satisfaction; it delivers
  - what the customer needs
  - when the customer needs it
- XP emphasises team work

- XP is fun

Probleme klassischer Vorgehensmodelle (2)

- Kundensicht
  - Kontraktbasiertheit / mangelnde Flexibilität
  - Aufwendige Einarbeitung in komplexes Produkt, das sehr spät zur Verfügung steht
  - Dokumentationsleichen

The solution: Feature-Driven Development

- How about delivering frequent, tangible, working results?
- Deliver results that are “useful in the eyes of the client”
- Use a feature as the smallest divisible task
- Attack very small block of client-valued functionality
- Group blocks into business-related sets
- Focus on delivering results every two weeks
- Track and report progress by feature progress

Defining features using FDD

- A feature is a client-valued function that can be implemented in two weeks
- Start with an informal list gathered from:
  - Conversations with domain members
  - Current documentation
- Build a detailed list after developing the overall model

Defining features using FDD (cont.)

Building an informal feature list helps to:

- Bring domain members together to talk
- Increase developer's understanding of the domain
- Fosters creativity and innovation
- Encourages exploring several questions:
  - What could be done?
  - What might be done?
  - What could make a real difference?
- Leads to the discovery of features that add significant business value

Roles within FDD (cont.)

Chief Programmer

- FDD requires someone to lead the FDD processes
- Should lead by example and mentor others
- Should be significantly more productive than other team members
- Adding programmers usually slows projects down, but adding an in-parallel chief programmer can accelerate a project

Roles within FDD (cont.)

Class Owner

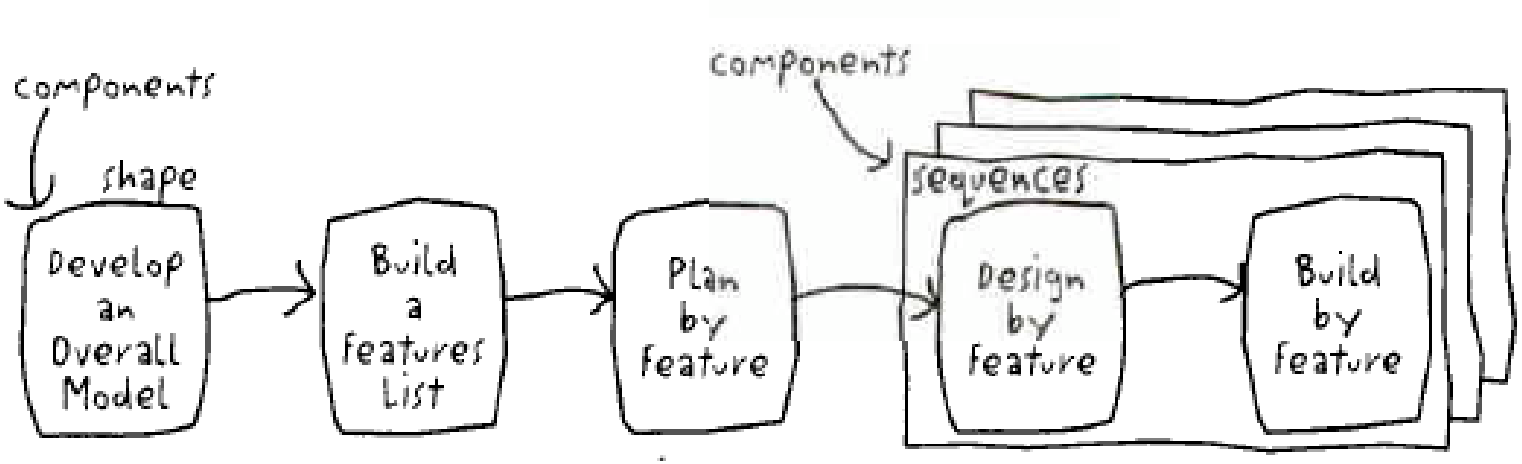
- Responsible for design and implementation of a class
- Developers gain a sense of “pride of ownership”
- Provides local consistency to a class
- The norm is one class to one class owner

Roles within FDD (cont.)

Feature team

- Features are assigned to a chief programmer
- The CP identifies class owners
- Together the CP and class owners form a temporary feature team
- Interactions are primarily between the CP and the class owners
- This ensures on-going mentoring and promotes uniformity of design and implementation

The five FDD processes



Pg. 190, Java Modeling in Color with UML

- Develop an overall model
- Build a detailed, prioritized features list
- Plan by feature
- Design by featuren (DBF)
- Build by feature (BBF)

The five FDD processes (cont.)

Why use a process?

- Lightweight process can help a team deliver results
- With larger projects, repeatable success is achievable
- New staff require shorter ramp-up time
- Focus on high-payoff results

What to avoid in a process

- Process for process' sake or "process pride"
- Process over-specification

The five FDD processes (cont.)

Develop an overall model

- Domain members and developers work together with chief architect
- Domain members present a high-level walkthrough
- Everyone works to develop a skeletal model
- Small pieces are presented by domain members in more details
- Sub-teams create a more detailed model which is merged into the skeletal model producing an overall model

The five FDD processes (cont.)

Build a detailed, prioritized features list

- The development team identifies features
- Features are grouped hierarchically
- A priority is assigned to each feature
- Each feature is weighted by importance
- Smaller teams can tackle specialized feature areas
- Domain member participation is welcome but not required

The five FDD processes (cont.)

Play by feature

- Using the features list project manager, development manager, and chief programmers establish milestones
- These milestones mark each design by feature and build by feature iterations

The five FDD processes (cont.)

Design by feature

- Chief programmer takes a feature iteration back to his group, now a feature team
- S/he identifies classes and assigns class owners
- The feature team works out a detailed sequence diagram
- Class owners prototype classes
- Team conducts a design inspection

The five FDD processes (cont.)

Build by feature

- Using the design by feature artifacts, each class owner implements their class methods
- Class owner implements class-level testing
- Feature team inspects the code
- After classes pass inspection the code is check into the version control system
- When all classes are checked in the chief programmer promotes the code to the build process

Comparing FDD to XP (cont.)

Metaphor and model

- User stories are replaced by domain walkthroughs
- Tasks are replaced by features
- Development of an overall domain object model
- Reduces the amount of refactoring required
- Gives more time to adding new features (from the features list)

Comparing FDD to XP (cont.)

Collective Ownership or Class Ownership

- Collective ownership usually degrades into “non-ownership”
- Any classes needing updates are members of the feature team
- Overly complex code won’t pass a code inspection
- Associated class owners work with other class owners which helps spread knowledge of the system

Comparing FDD to XP (cont.)

Inspections and Pair Programming

- No reason why programmers can't pair up inside of their feature teams
- FDD promotes code inspections over pair programming
- Allows fresh eyes to look at the code
- Chief programmer will help ensure use of best practices
- Change of pace for developers

Comparing FDD to XP (cont.)

Testing

- Incorporated into build by feature process
- No mechanism for formal unit testing
- Chief programmer left to do what is appropriate
- Unit testing can be used depending on technology and resources
- Producing completely isolated test may be difficult in a reasonable amount of time

Comparing FDD to XP (cont.)

Reporting

- Part of the Tracking by Feature process in FDD
- Low-overhead highly accurate means of measuring progress
- Provides data to create practical progress charts and graphs

Generalization: Agile Methods

- XP
  - The Planning Game, Small Releases, Metaphor, Simple Design, Testing, Refactoring, Pair Programming, Collective Ownership, Continuous Integration, 40-hour Week, On-site Customer, Coding Standards

SCRUM

What is Scrum? (from controlchaos.com)

- Scrum is an iterative, incremental process for developing any product or managing any work. It produces a potentially shippable set of functionality at the end of every iteration. It's attributes are:
- Scrum is an agile process to manage and control development work.
- Scrum is a wrapper for existing engineering practices.
- Scrum is a team-based approach to iteratively, incrementally develop systems and products when requirements are rapidly changing
- Scrum is a process that controls the chaos of conflicting interests and needs.
- Scrum is a way to improve communications and maximize co-operation.
- Scrum is a way to detect and cause the removal of anything that gets in the way of developing and delivering products.
- Scrum is a way to maximize productivity.
- Scrum is scalable from single projects to entire organizations. Scrum has controlled and organized development and implementation for multiple interrelated products and projects with over a thousand developers and implementers.
- Scrum is a way for everyone to feel good about their job, their contributions, and that they have done the very best they possibly could.

DSDM (Dynamic System Development Method)

1. **Active user involvement** is imperative.
2. The **team must be empowered** to make decisions.
3. The focus is on **frequent delivery of products**.
4. **Fitness for business purpose** is the essential criterion for acceptance of deliverables.
5. **Iterative and incremental development** is necessary to converge on an accurate business solution.
6. All **changes** during development **are reversible**.
7. **Requirements** are baselined at a **high level**.
8. **Testing is integrated** throughout the life-cycle.
9. **Collaboration and cooperation** between all stakeholders **is essential**.

From http://www.dsdm.org/tour/principles.asp

Are Agile Methods People Friendly?

	XP	SCRUM	DSDM
Achievement	The Planning Game, Small Releases, Simple Design, Testing, Refactoring, Pair Programming, Collective ownership, Continuous integration, On-site customer,	controls the chaos of conflicting interests and needs; detect and cause the removal of anything that gets in the way of developing and delivering products	Active user involvement, team empowered to make decisions, focus is on frequent delivery, Iterative and incremental development, all changes during development are reversible, testing is integrated
Possibility of growth	Refactoring, Pair Programming, Collective ownership,		team empowered to make decisions, all changes during development are reversible
Work itself	On-site customer, Refactoring,		Active user involvement , team empowered to make decisions, all changes during development are reversible
Recognition	Pair Programming, Collective ownership, On-site customer,		Active user involvement

Are Agile Methods People Friendly?

	XP	SCRUM	DSDM
Advancement	Pair Programming, Collective ownership, On-site customer,		Active user involvement
Supervision technical	Refactoring, Pair Programming, Collective ownership,		all changes during development are reversible
Responsibility	The Planning Game, Testing, Collective ownership, 40 hour work week, On-site customer,		Active user involvement, testing is integrated
Relations peers	Refactoring, Pair Programming, Collective ownership, 40 hour work week,	improve communications and maximize co-operation	all changes during development are reversible

Are Agile Methods People Friendly?

	XP	SCRUM	DSDM
Relations subordinate			
Salary			
Personal life	40 hour work week,		
Relations superior	40 hour work week,		

Are Agile Methods People Friendly?

	XP	SCRUM	DSDM	AM
Job security	Temporary +	Temporary +	Temporary +	Temporary +
Status	On-site customer,			
Company policy and administration				
Working conditions	40 hour work week,			

Agile vs. Tayloristic Methods

- Agile methods
  - Human-centric
  - Tacit knowledge sharing
  - Code-centric
  - Replace documentation by face-to-face communication
  - Generalists
  - Plan and correct
  - Customer-focused
- Tayloristic methods (plan-driven, traditional, heavyweight)
  - Process-centric
  - Explicit knowledge
  - Documentation-centric
  - Role specialization
  - Plan and control
  - Contract-focused