

Vorlesung "Software-Engineering"

Prof. Ralf Möller, TUHH, Arbeitsbereich STS

■ Vorige Vorlesung

- Testen
- Konstruktive Qualitätssicherung

■ Heute

- Probleme klassischer Vorgehensmodelle
- Extreme Programming (XP)

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 klassischer 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

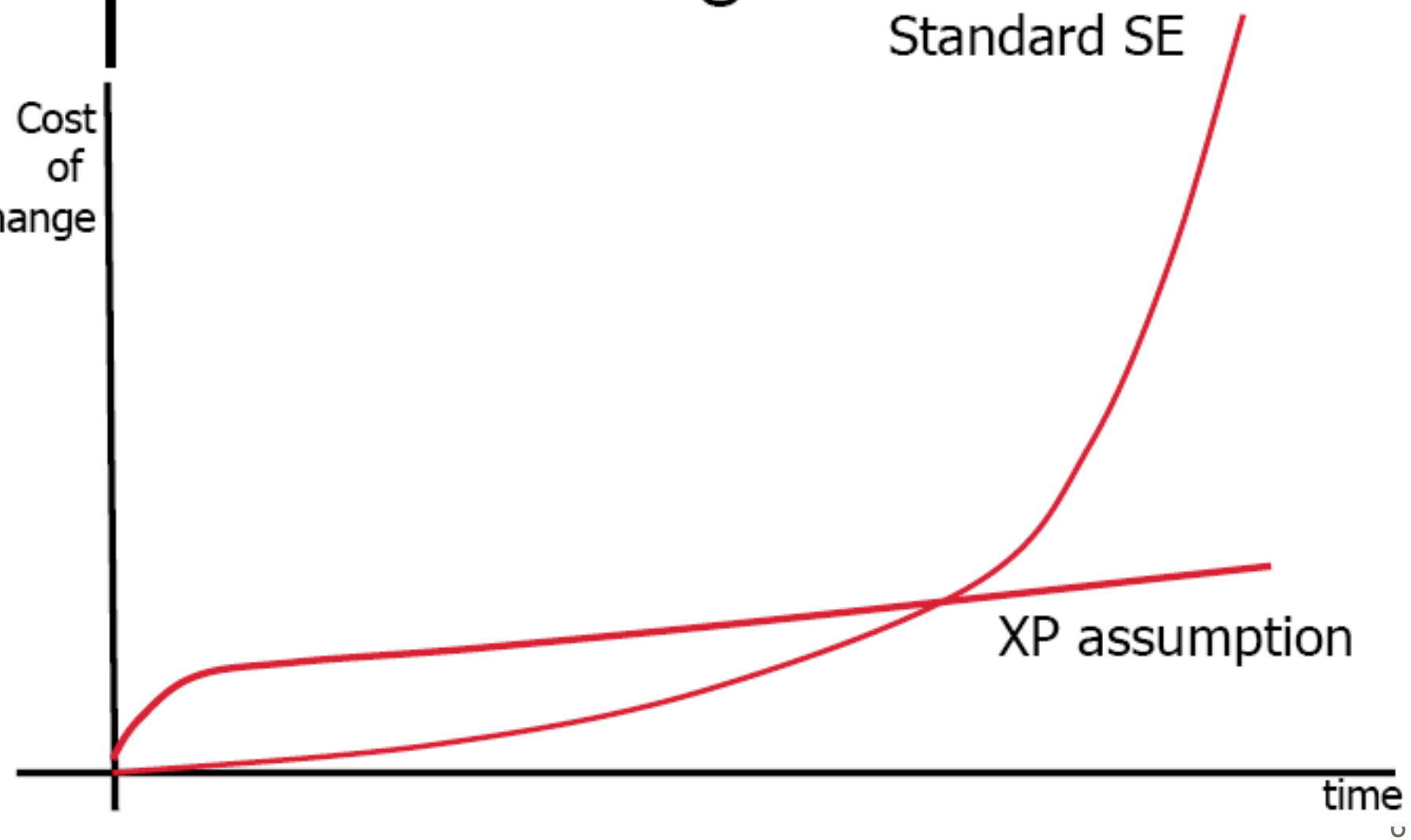
Practices

- 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



Cost of change



Standard SE

XP assumption

time

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 / Tasks

- 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(s) 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 programming 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
- However, most of XP focuses developer view

Probleme klassischer Vorgehensmodelle (2)

■ Kundensicht

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

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