Course "Softwareprozesse"

**Agile Technical Practices:**

**eXtreme Programming (XP), Part I**

Lutz Prechelt
Freie Universität Berlin, Institut für Informatik

- Structure of agile methods
  - values, roles, technical practices, mgmt. practices
- eXtreme Programming (XP)
  - XP1 vs. XP2 vs. Jeffries
  - Values, roles
  - Management practices
- Technical practices:
  - Continuous Integration
    - Ten-minute Build, Feat. Toggle, C. Delivery, C. Deployment
- Test-first Programming
  - Testing, TDD, ATDD
Learning objectives
(for parts I & II together)

- Understand
  - the structure of methods,
  - the role of practices, and
  - the difference between management practices and technical practices

- Roughly understand the practices that make up XP and how they play together
  - including pros and cons (including some research results)

- Roughly understand when to and when not to use XP
XP sources

• Kent Beck, Cynthia Andres: "Extreme Programming Explained: Embrace Change", Addison-Wesley, 2004
  • 2nd edition (XP, XP2); complete rewrite of 1999 1st edition
    • See article on 1st edition (XP1)
    • Different set of practices!
  • Worth reading!

• Ron Jeffries (xprogramming.com) uses a still different mix
  • Beck and Jeffries are the co-inventors of XP

• See also other books and articles, c2.com XP roadmap, Agile Alliance summary, ...
Methods vs. practices

• "method": a systematic procedure for attaining something
  • states how to do it (e.g. a cooking recipe, an algorithm)
  • Most so-called "methods" are hardly methods
    • e.g. Scrum states what to achieve, but rarely how to do it

• "practice": "[What] one does as a habitual or customary action or act"
  • https://wiki.c2.com/?ApproachesMethodsAndPractices
  • XP explicitly consists of values, principles, and practices
Methods vs. practices (2)

• Methods have a positivist touch
  • "This is how to do it!"
  • "If you don't do it like this, you are a fool/outlaw"
    • There is a lot of such thinking in would-be agile circles

• Practices have a humanist touch
  • "Here is something we tend to do because it is a Good Idea"
  • "Feel free to deviate if needed. If really needed."

• XP, Chapter 3:
  • "Practices are evidence of values."; "Practices are clear"
  • "Bridging the gap between values and practices are principles. [...] Principles are domain-specific guidelines for life."
    • Many so-called methods would better be called principles or sets of principles
    • Scrum's Sprint Review & Retrospective can be considered principles.
XP values

- Communication
- Simplicity
  - "Simplicity is the most intensely intellectual of the XP values. To make a system simple enough to gracefully solve only today's problem is hard work."
- Feedback
  - "we use feedback to get closer and closer to our goals."
- Courage
  - "Courage is effective action in the face of fear."
  - "Sometimes [...] manifests as a bias to action. [...] Sometimes courage manifests as patience."
- Respect
  - "I am important and so are you."

(Scrum's values are Commitment, Focus, Openness, Respect, and Courage but Scrum has almost no explanation what they mean. XP does.)
XP principles

"Principles are domain-specific guidelines for life."

- Too many to discuss here:
  - Humanity, Economics, Mutual benefit, Self-similarity, Improvement, Diversity, Reflection, Flow, Opportunity, Redundancy, Failure, Quality, Baby steps, Accepted responsibility
  - Many we already know, e.g. Humanity, Economics, Improvement, Diversity, Reflection, Flow, Quality

- Some are really interesting:
  - Opportunity: "see problems as opportunities for change."
  - Failure: "If you're having trouble succeeding, fail. [...] Isn't failure waste? No, not if it imparts knowledge."
  - Baby steps: "What's the least you could do that is recognizably in the right direction? [...] [The] overhead of small steps is much less than when a team wastefully recoils from aborted big changes."
    - Could almost be considered a technical practice

Lutz Prechelt, prechelt@inf.fu-berlin.de
"To make agile work, you need solid technical practices. A lot of agile education under-emphasizes these, but if you skimp on this you won't gain the productivity and responsiveness benefits that agile development can give you (stranding you at level 1 of the agile fluency model.)

This is one of the reasons that I still think that Extreme Programming is the most valuable of the named agile methods as a core and starting point."

http://martinfowler.com/agile.html
### Practices of XP, XP2, Jeffries' XP
(furthermore, XP2 has 11 "Corollary Practices")

<table>
<thead>
<tr>
<th><strong>XP1 practices (&quot;traditional&quot;)</strong></th>
<th><strong>XP2 practices (&quot;evolutionary&quot;)</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>1. The Planning Game</td>
<td>M ✓ 1. Stories</td>
</tr>
<tr>
<td>2. Small Releases</td>
<td>M ✓ 2. Weekly Cycle</td>
</tr>
<tr>
<td>3. 40-Hour Week</td>
<td>M ✓ 3. Quarterly Cycle</td>
</tr>
<tr>
<td>4. On-Site Customer</td>
<td>M ✓ 4. Energized Work</td>
</tr>
<tr>
<td>5. Pair Programming</td>
<td>T 5. Slack</td>
</tr>
<tr>
<td>6. Collective Ownership</td>
<td>T 6. Whole Team</td>
</tr>
<tr>
<td>7. Metaphor</td>
<td>T 7. Sit Together</td>
</tr>
<tr>
<td>8. Simple Design</td>
<td>T 8. Informative Workspace</td>
</tr>
<tr>
<td>10. Testing</td>
<td>T 10. Incremental Design</td>
</tr>
<tr>
<td>11. Continuous Integration</td>
<td>T 11. Test-First Programming</td>
</tr>
<tr>
<td>12. Coding Standards</td>
<td>T 12. Continuous Integration</td>
</tr>
<tr>
<td></td>
<td>13. Ten-Minute Build</td>
</tr>
</tbody>
</table>

**M**: Mgmt, **T**: Technical

**J**: Jeffries' additional practice:
- Customer tests

Lutz Prechelt, prechelt@inf.fu-berlin.de
XP practices: XP2, XP1, XP2 "corollary" (optional)

Note: Some connections are missing

Graphic: Stefan Roock

Lutz Prechelt, prechelt@inf.fu-berlin.de
Ron Jeffries' view: core, infrastructure, customer interface

XP Practices

We'll pay a lot of attention to the core

https://ronjeffries.com/xprog/what-is-extreme-programming/
Kent Beck's stance on practices

• Chapter 6 "Practices":
  • "Applying a practice is a choice.
  • I think the practices make programming more effective.
  • This is a collection of practices that work and work even better together. They have been used before.
  • Experiment with XP using these practices as your hypotheses. For example, let’s try deploying more frequently and see if that helps."

• Chapter 7 "Primary Practices":
  • "Practices are theories, predictions."
    • About what behaviors are useful and what they achieve.
    • Such predictions can be wrong, given a team or situation!
"Integrate and test changes after no more than a couple of hours."

- An automated process (1) builds the system,
  (2) runs the automated tests, (3) logs results

- This **build** represents the project state
  - The build must be fully functional at almost any time
    - A build that remains broken for some time is an indicator of bad project health
    - Teams without a healthy CI cannot be agile

- Version-management branches make CI difficult
  - How many different builds are you willing to run?
  - How will developers understand which ones to pay attention to?
  - How often are you willing to modify your CI setup?
Continuous Integration: Feature Toggles

- Use feature toggles instead and run two builds:
  - Production-like settings
  - All features "on"

- Feature-toggle practices [MahDreWil21]:
  - practitioner survey & literature study
  - Have a toggle mgmt system:
    - consciously decide each toggle introduction
    - metadata (documentation: owner, status, ...)
    - naming convention, default value
    - change log
  - Group toggles, manage dependencies
  - Expiration date (e.g. as an automated test)
    - limit number of toggles

Lutz Prechelt, prechelt@inf.fu-berlin.de
XP2: Ten-minute build

- "Automatically build the whole system and run all of the tests in ten minutes."
  - If it takes longer, it will be used less ➔ reduced feedback
    - making repairs more costly
  - So when the build gets slower, optimize it, e.g.
    - find a tool that runs only those tests that execute changed code
    - but make sure to run tests relying on external services
  - For large systems, modularize more
  - Replace individual integration tests if they are slow

- GUI-based system tests make this difficult
  - Why do you need so many of them?
    - ➔ Incremental Design, Test-First Programming
Beyond CI: Continuous Delivery (CD)

• "Continuous Delivery is a SW development discipline where you build software in such a way that the software can be released to production at any time." [Fowler13]
  - Having a CI is not enough! One needs to
  - prioritize keeping-it-deployable over
    - working on new features
    - long-running restructurings;
  - have a DevOps culture (no silos, autonomy) [Wilsenach15];
  - have fully automated "push-button" deployment

• Kanban demands Continuous Delivery; Scrum & XP do not

• Beware of confusing Continuous Delivery (CD) with Continuous Deployment (CD):
Beyond CI: Continuous Deployment (CD)

- When a build is successful, it will automatically and immediately be deployed to the production system
  - So effectively iterations become extremely short
  - Only possible for web-based and similar systems
  - Precondition: Continuous Delivery

- An ambitious goal!
  - high risk of breaking something
  - some top companies did or do this (e.g. Amazon, Facebook)

https://www.agilealliance.org/glossary/continuous-deployment/
XP1: Testing, J: Test-driven development
XP2: Test-First Programming

- "Write a failing automated test before changing any code."
  - Write test; see it fail; write code; see test succeed; repeat
- "Test-first programming addresses many problems at once:
  - Scope creep [by having to stay focused]
  - [Low] Coupling and cohesion [or else testing is difficult]
  - Trust [...] [G]ive your teammates a reason to trust you
  - Rhythm: [...] [Develop in] a natural and efficient rhythm --
    - test, code, refactor, test, code, refactor."
- "As your experience grows, you’ll be able to squeeze more and more reassurance into these tests."

- Beck does not discuss granularity
  - but implies a fine granularity:
    - "Because of their limited scope, these tests tend to run very fast.
      - You can run thousands of them as part of the Ten-Minute Build."
Test-first programming: Discussion

• Oft-claimed advantages:
  • Clarifies the requirements for the element before coding it
  • Defines the interface
  • "First": helps keeping up the discipline
  • Provides rapid and constant feedback
  • Thus allows courage during refactoring

• Suitability depends on a suitable granularity of "changing any code"
  • A too-small granularity may be exaggerated
    • Some people insist on iterations of ~1 minute length
Test-first programming: How much?

Kent Beck (on stackoverflow 2008):
• "I get paid for code that works, not for tests,
  • so my philosophy is to test as little as possible to reach a given level of confidence [...].
  • *If I don't typically make a kind of mistake [...]*, I don't test for it."

• David Heinemeier Hansson (author of Ruby on Rails, 2012)
  • "Testing just what's useful takes nuance, experience, and dozens of fine-grained heuristics."

⇒ A difficult question!
Test-first programming/TDD: Personal experience?

If you do serious SW development:

• Do you use thorough automated testing?
  • Often? Nearly always?

• Did you ever try test-first programming/TDD?
  • Did you try to make it a habit?
  • Pros of it? Cons?
  • When and where?
    • Logic? GUI? Integration?
    • New vs. existing code?

• Do your colleagues use it?

• How good is the test suite overall?
  • Code coverage?
  • How much confidence does it provide?
Research: What limits industrial TDD adoption?

CauSunPun11: "Factors limiting industrial adoption of TDD: a systematic review"

- based on 48 empirical studies on TDD, mostly case studies or experiments
- Dev. time often increased
  - sometimes decreased
- Many industrial devs lack TDD knowledge
  - Or generally lack skill to find good test cases
- Sometimes architecture problems
  - (the article is vague here)

- Technical/tool problems
  - esp. for GUI testing, network testing
- Lack of discipline
  - e.g. time pressure, no obvious benefits
  - two studies found low TDD correlated with low quality
    - in orgs that prefered TDD
- Legacy code
  - (near-)lack of test suite
    - and testable structure

Successfully using TDD is difficult!

Lutz Prechelt, prechelt@inf.fu-berlin.de
Research: Effects from TDD

- BisSerFig16: "The effects of TDD on internal quality, ext. qual. & productivity: A systematic review"
  - based on 27 studies: 57% using experiments, 32% using a case study
  - comparison to test-last

- Trends:
  - Academic environments: Productivity increases
  - Industrial environments: Productivity decreases
  - 76% of studies report better internal SW quality
    - e.g. lower coupling
  - 88% report better external SW quality (reliability)

- Conclusion:
  - TDD tends to help, but is not free.

Lutz Prechelt, prechelt@inf.fu-berlin.de
J: Customer Tests (ATDD)

- Write automated tests at the story-level
  - testing relevant, user-visible, valuable functionality directly
  - ideally in a form the end user can read (for validation).
  - They then serve as always up-to-date documentation.
    - Very useful for user support.
  - a.k.a. ATDD: Acceptance-Test-Driven Development

- These add confidence beyond what unit tests and integration tests can provide
  - balance with the unit and integration tests, limit redundancy
  - write more of them if you often break stories

https://ronjeffries.com/xprog/xpmag/problems-with-acceptance-testing/
https://ronjeffries.com/xprog/blog/automating-story-tests/

Lutz Prechelt, prechelt@inf.fu-berlin.de
Summary

- The "technical excellence" so important for Agile requires technical practices
  - Scrum and Kanban do not offer any; XP does (based on values)

- Continuous Integration, Ten-Minute Build
  - are important foundations for agile work
  - are the basis for Continuous Delivery (Kanban), let alone Continuous Deployment

- Thorough automated tests are super important
  - for creating the confidence required for
    - making changes and
    - keeping the design structure intact
  - and a Test-First workstyle and Customer Tests can be helpful for creating them
Thank you!