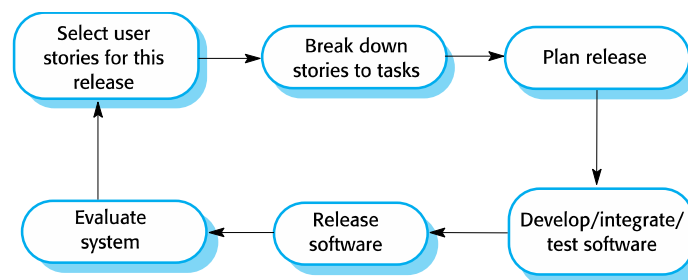


Extreme programming

- An influential agile method, developed in the late 1990s, that introduced a range of agile development techniques.
- Some - though not all – XP practices are in common use today.
- Extreme Programming (XP) takes an 'extreme' approach to iterative development.
 - New versions may be built several times per day;
 - Increments are delivered to customers every 2 weeks;
 - All tests must be run for every build and the build is only accepted if tests run successfully.

1

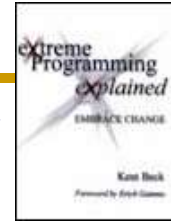
The extreme programming release cycle



2

Extreme programming

- Set of SE practices that produce **high-quality software** with **limited effort**
- Light-weight software development process
 - Replaces documentation with communication
 - Focuses on source code and testing
 - **Controversial** – “Hacking”?
 - Strong **productivity** improvements
- Developed by industry practitioners
 - “..proven at cost conscious companies like Bayerische Landesbank, Credit Swiss Life, DaimlerChrysler, First Union National Bank, Ford Motor Company and UBS.”
- XP Web Site: <http://www.extremeprogramming.org>



XP Challenges Assumptions

- XP says that **analogies** between software engineering and **other engineering domains** are **false**:
 - software customers' requirements change more frequently;
 - our products can be changed more easily;
 - the ratio of **design cost:build cost** is much higher;
 - if we consider coding as “design” and compile-link as “build”:
 - the “build” task is so quick and cheap it should be considered instant and free,
 - almost all software development is “design”.
- The design meets known existing requirements, not all possible future functionality

Fundamentals of XP

- Distinguish between decisions made by business stakeholders and developers
- Simplistic – keep design as simple as possible
“design for today not for tomorrow”
- Write automated test code before writing production code and keep all tests running
- Pair programming
- Very short iterations with fast delivery

Why is XP controversial?

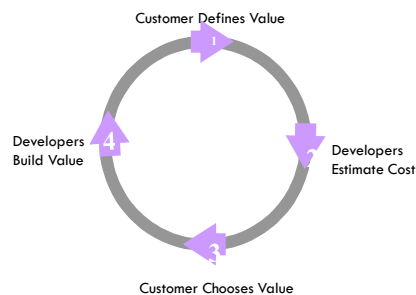
- No specialists
 - Every programmer participates in architecture, design, test, integration
- No up-front detailed analysis and design
- No up-front development of infrastructure
- Not much writing of design & implementation documentation beside tests and code

When can XP be used?

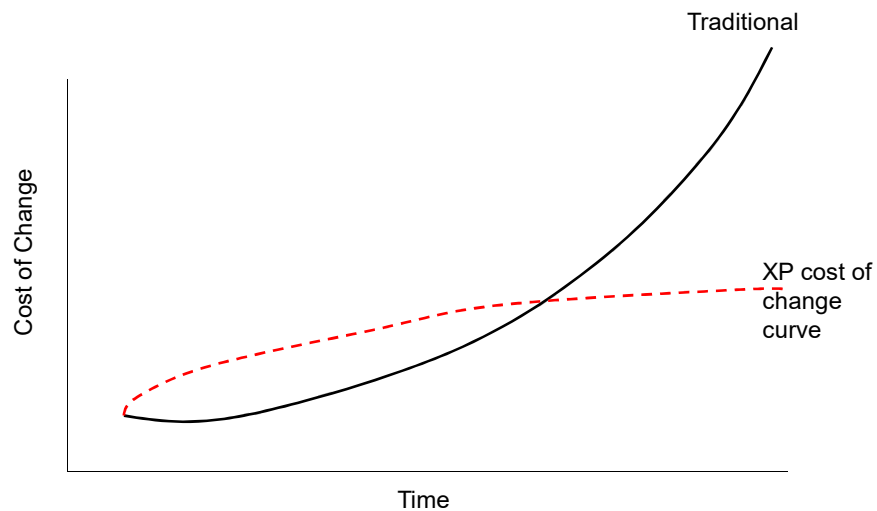
- Small projects:
 - 5-10 developers, maybe 20
- Developer and customer representative are co-located
- Problems:
 - Point-and-go culture
 - Testing takes hours to execute

XP Definitions

- Kent Beck's idea of turning the knobs on all the best practices up to 10.
- Optimizing the "**Circle of Life**" by hitting the sweet-spot of practices that self-reinforce and become more than the sum of the parts (synergize).



XP Cost of Change Curve



The Four XP Values

- **Simplicity**
 - Simplest thing that could possibly work
 - YAGNI
- **Communication**
 - Developers
 - Users
 - Customers
 - Testers
 - Code
- **Feedback**
 - Testing
 - Experimenting
 - Delivering
- **Courage**
 - Trust
 - History

Twelve XP Practices

- | | |
|---------------------|---------------------------|
| 1. Planning Game * | 7. Collective Ownership |
| 2. Short Releases * | 8. Continuous Integration |
| 3. Simple Design * | 9. On-site Customer * |
| 4. Testing * | 10. Sustainable Pace * |
| 5. Refactoring | 11. Metaphor |
| 6. Pair Programming | 12. Coding Standards * |

- Many of the practices actually existed, in one form or another, prior to the advent of XP (*)

Influential XP practices

- Extreme programming has a technical focus and is not easy to integrate with management practice in most organizations.
- Consequently, while agile development uses practices from XP, the method as originally defined is not widely used.
- **Key practices**
 - User stories for specification
 - Refactoring
 - Test-first development
 - Pair programming
 - CI

13

The Planning Game

- Recognition
 - You can't know everything when you start
 - What a realization!
 - Customers will learn about what they want by looking at what you've done so far
 - Developers will learn about the domain and technology as the project progresses.
 - Developers will learn what questions to ask once they start
 - Early on, they don't know what they don't know

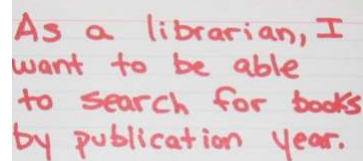
1. Planning Game

- **Release Planning:**
 - Define and estimate higher-level features down to about 5-10 days effort each.
 - Customer lays features in fixed-length iteration schedule.
- **Iteration Planning**
 - Same, but to 3 or less days effort & detailed story cards within next iteration.
- Simple to steer project towards success.

The Planning Game

- **User Story**

- Written on a card
 - “chunk of functionality that is coherent in some way to the customer”
- Given **priority/value** by the customers based on business value
- Given **estimate** (1, 2, 3 weeks) by the developers
- Given **risk** value by developers
 - Doing risky items first mitigates risk
- The card is a commitment to talk more later.



As a librarian, I
want to be able
to search for books
by publication year.

- **Example**

- Employees who are sick more than 3 days go on DAP (Disability Absence Plan). They are paid from their full pay for 190 working days, and then 70% pay up through 270 days. DAP euros paid must be kept separate from regular pay euros, for accounting purposes.

User Story Examples

A user wants access to the system, so they find a system administrator, who enters in the user's First Name, Last Name, Middle Initial, E-Mail Address, Username (unique), and Phone Number.

Risk: Low

Cost: 2 points

The user must be able to search for a book.

Risk: High

Cost: (too large!)

The user must be able to search for a book by Title, and display the results as a list.

Risk: Med.

Cost: 1 point

The user must be able to search for a book by Category, and display the results as a list.

Risk: Med.

Cost: 2 points

2. Short Releases

- **Deliver business value early and often**
- Do not slip iteration release dates
 - adjust scope within an iteration, never time or quality
- Small, stable teams are predictable in short time-frames
- De-scope as opposed to Delay!

3. Simple Design

- XP Mantra
 - “The simplest thing that could possibly work”**
- Meet current, minimum business requirements only
- Avoid anticipatory design
- YAGNI – You Aren’t Going to Need It

4. Testing

- Its all about automation.
- Automated unit tests for every entity.
- Automated acceptance tests for every story / requirement.
- All unit tests pass 100% before checking in a feature.

Testing

- **Unit Tests**
 - Code Unit Test First (in small increments)
 - When developers go to release new code, they run all the unit tests, not just theirs, on the integration machine.
 - The tests must run at 100% before checking in feature
 - If any test fails, they figure out why and fix the problem.
 - The problem certainly resides in something they did ... since they know the tests ran at 100% the last time anything was released.

Testing

- **Acceptance Tests**

- Test cases “extracted from” customer
- Test system end-to-end
- Tells the customer and the developers if the system has the features it is supposed to have
- Don’t have to run at 100%
- Progress used to measure “Project Velocity”
 - What % of the customer’s acceptance test cases run?

Testing

- Automating software testing

- Manual software testing is time consuming
- Software testing has to be repeated after every change (regression testing)
- Write test drivers that can run automatically and produce a test report

- Junit testing

- A small testing framework written in Java.
- A series of extensible classes that do much of the testing grunt work for us. (i.e. counting and reporting errors and failed tests, running tests in batch, etc.)
- Very handy for Extreme Testing...
- Developed by Kent Beck and Erich Gamma

- Many other unit testing frameworks exist, e.g. pyunit.

Test automation

- Test automation means that tests are written as executable components before the task is implemented
 - These testing components should be stand-alone, should simulate the submission of input to be tested and should check that the result meets the output specification.
 - An automated test framework (e.g. Junit) is a system that makes it easy to write executable tests and submit a set of tests for execution.
- As testing is automated, there is always a set of tests that can be quickly and easily executed
 - Whenever any functionality is added to the system, the tests can be run and problems that the new code has introduced can be caught immediately.

24

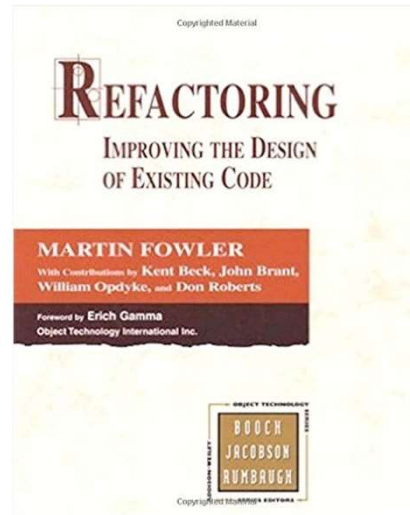
Problems with test-first development

- Programmers prefer programming to testing and sometimes they take short cuts when writing tests.
 - For example, they may write incomplete tests that do not check for all possible exceptions that may occur.
- Some tests can be very difficult to write incrementally. For example, in a complex user interface, it is often difficult to write unit tests for the code that implements the 'display logic' and workflow between screens.
- It difficult to judge the completeness of a set of tests. Although you may have a lot of system tests, your test set may not provide complete coverage.

25

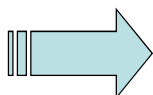
5. Refactoring

- Refactoring: changing internal structure without changing external behavior
- When change is hard, refactor to allow change to be easy, testing as you go, then add change.



Simplicity leads to Refactoring

- Refactoring = Changing the code without changing its functionality
- Remove duplication(s)
- Goal: make code easier to maintain (keep in simplest form)



investment into the future

Refactoring book by Fowler

<http://www.refactoring.com>

Simplicity leads to Refactoring

- **Why refactor?**

- To improve the design of software
- To make code easier to understand
- To help find bugs
- As a result
 - (future) coding becomes faster
 - outside source code documentation less required

Simplicity leads to Refactoring

- **When do you refactor?**

- When the code “smells bad”
 - Repeating code
 - Code difficult to understand
 - Long methods / functions
 - ...

Examples of refactoring

- Re-organization of a class hierarchy to remove duplicate code.
- Tidying up and renaming attributes and methods to make them easier to understand.
- The replacement of inline code with calls to methods that have been included in a program library.

30

6. Pair Programming

- **Two heads are better than one**, especially in an open lab environment (co-location)
- Earliest possible code inspections
- Earliest possible brainstorming
- Peer pressure reinforces discipline



Pair Programming

- Traditional specialization leads to queues (you will block on the specialist at some point)
- the 'Driver' implements, focusing on the tactical.
- The 'Navigator' is more objective (strategic) – asks 'why' and provides explanations
- Healthy pairings have communications every 45 to 60 seconds.
- Pairs should switch roles often & can break off for low complexity tasks
- Pairing is intense (suggest a break every 2 hours)

Pair Programming

- Knowledge transfer amongst the team is also a major advantage when using pair programming
 - Both programmers in the pair are familiar with the code and have either written the code or has been actively involved as the programmer watching the code generation.
- Development costs do not double
- Continual training...

Is Pair Programming Costly?

- **Cost of pair programming**
- Williams, Laurie, Kessler, Robert R., Cunningham, Ward, and Jeffries, Ron, [Strengthening the Case for Pair-Programming](#), IEEE Software, July/Aug 2000 .
 - University study with 41 students
 - Higher quality code
 - Test cases passed individuals: 73.4%-78.1%
 - Test cases passed pairs: 86.4%-94.4%
 - Pairs completed assignments 40-50% faster (average 15% higher costs) **Why might this be the case?**
 - Pair programming preferred by students (85%)

7. Collective Ownership

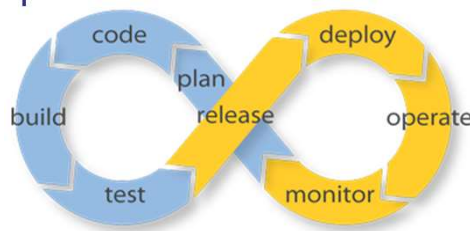
- Interchangeable programmers
- Team can go at full speed
- Can change anything, anytime, without delay

Collective Code Ownership



8. Continuous Integration

- Avoids “versionitis” by keeping all the programmers on the same page
- Integration problems smaller, taken one at a time
- Reduces risks associated with the traditional integration phase



9. On-site Customer

- Customer/User liaisons are team-members
- Available for priorities, clarifications, to answer detailed questions
- Reduces programmer assumptions about business value
- Shows stakeholders what they pay for, and why
- **On-site customer ensures:**
 - Developers don't have to wait for decisions
 - Face to face communication minimizes the chances of misunderstanding
 - Remember, the original user story was a commitment for a later conversation . . .

10. Sustainable Pace

- Tired programmers make more mistakes
- Better to stay fresh, healthy, positive, and effective
- XP is for the average programmer, for the long run

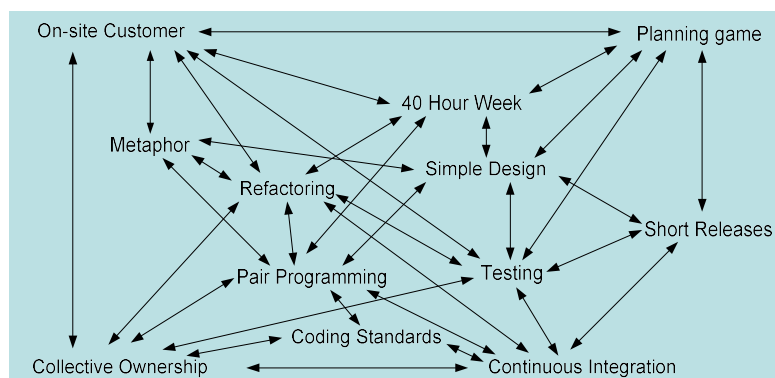
11. Metaphor

- Use a “system of names”
- Use a common system description
- Helps communicate with customers, users, stakeholders, and programmers

12. Coding Standards

- All programmers write the same way
- Rules for how things communicate with each other
- Guidelines for what and how to document

XP Practices Support Each Other



Source: Beck, Extreme Programming Explained: Embrace Change, 1999

Some advantages of XP

- Promotes team cooperation and support (pair programming)
- Strong support for changing / emerging requirements
- Can develop discipline and team ethic for excellence
- Can improve quality (afterall, code inspections are happening as the code is written)
- Recognises that the requirements cannot be fully captured at the beginning and that customers need to continue to be heavily involved throughout the project (co-responsibility)
- Provides for continual training and knowledge transfer
- Avoids over specialisation
- Test-first is cost effective, so is early inspecting

Some limitations of XP

- Can the customer realistically be on-site?
- Co-location of team members required
- Question marks regarding scalability of the process: Small teams → small projects
- Distributed XP: Can we transfer the productivity benefits of XP to a distributed environment?
 - Communication replaces documentation (KT? Training? Selling company?)
 - Open-source projects
- Virtual teams
 - Software development is more and more distributed
 - Size of projects, Scarce local resources
 - Outsourcing of tasks is often financially beneficial
 - Open source success stories
- Extreme Programming Considered Harmful for Reliable Software Development <http://www.agilealliance.org/system/article/file/945/file.pdf>
- Is the concept that all people can be equally effective fundamentally flawed? *Premium* people debate...

Reflections on XP

- Not much evidence to suggest that XP continues to be in widespread use today.
- But as a set of practices, XP has had a profound impact on the software development process.
- Many of the practices used in combination in XP can be seen in active use in practice and in the descriptions of later agile methods.
- XP can therefore be considered to have catalysed a significant change in the approach to the software development process.