ETH zürich

Agile software development

Bertrand Meyer

Part D: agile practices

1: Meetings

2: Development

3: Release

4: Testing and quality

5: Management and others

Daily meeting

XP Scrum

Held every morning

Goal: set the day's work, in the broader context of the project

Time-limited, usually 15 minutes ("stand-up meeting")

Involves all team members, with special role for "committed" (over just "involved")

Focus:

- Defining commitments
- Uncovering impediments

Resolution will take place outside of meeting

Daily meeting: the three questions

The daily meeting requires every team member to answer three questions:

- What did you do yesterday?
- What will you do today?
- Are there any impediments in your way?

Planning meeting

At beginning of every sprint

Goal: define work for sprint

Outcome: Sprint Backlog, with time estimate for every task

8-hour time limit

- 1st half, product owner + team: prioritize product backlog
- 2nd half, team only: plan for Sprint, producing sprint backlog

Retrospective

Scrum

All team members reflect on past sprint

Make continuous process improvements

Two main questions:

- What went well?
- What could be improved?

3-hour time limit

Review meeting

Scrum

Review work:

- Completed
- Not completed

Present and demo completed work to stakeholders

Incomplete work cannot be demonstrated

4-hour time limit

Reflective improvement

Developers must take breaks from regular development to look for ways to improve the process

Iterations help with this by providing feedback on whether or not the current process is working

Agile Software Development

Bertrand Meyer

Part D: Practices
1: Meetings

What we have seen:

Agile development is characterized by a set of well-defined meetings

Most important is the Daily Meeting

In Scrum: planning and review meetings

Require adaptations for distributed teams

ETH zürich

Agile software development

Bertrand Meyer

Part D: agile practices

1: Meetings

2: Development

3: Release

4: Testing and quality

5: Management and others

Pair programming

XP

Source: Beck 2005

Two programmers sitting at one machine

Thinking out loud

Goals:

- Make thinking process explicit
- Keep each other on task
- Brainstorm refinements to system
- Clarify ideas
- Take initiative when other stuck, lowering frustration
- Hold each other accountable to team practices

Single code base

Maintain a single code base: avoid branching, even if permitted by configuration management system

Shared code

Agile methods reject code ownership in favor of code whose responsibility is shared by entire team

Rationale:

- Most non-trivial features extend across many layers in the application
- Code ownership creates unnecessary dependencies between team members and delays
- What counts is implemented features, not personal responsibility
- Avoid blame game
- Avoid specialization
- Minimize risk (team members leaving)

Leave optimization till last

XP

Source: Wallace 02

Wait until you have finished a story and run your tests before you try to optimize your work

Only then can you analyze what exactly it is that needs optimizing

Do not make work for yourself by trying to anticipate problems before they exist

Simple design

Produce the simplest design that works

Refactor as needed

Incremental design

XP

Source: Shore 08

Developers work in small steps, validating each before moving to the next
Three parts:

- Start by creating the simplest design that could possibly work
- Incrementally add to it as the needs of the software evolve
- Continuously improve design by reflecting on its strengths and weaknesses

"*When you first create a design element, be completely specific. Create a simple design that solves only the problem you face, no matter how easy it may seem to solve more general problems.*

"The ability to think in abstractions is often a sign of a good programmer. Coding for one specific scenario will seem strange, even unprofessional. Waiting to create abstractions will enable you to create designs that are simple and powerful. Do it anyway."

System metaphor

XP

Source: Tomayko 03

"A metaphor is meant to be agreed upon by all members of a project as a means of simply explaining the purpose of the project and thus guide the structure of the architecture"

Benefits:

- Communication, including between customers & developers
- Clarify project, explain functionality
- Favors simple design
- Helps find common vocabulary

Example metaphors

XP

Source: Tomayko 03

Project	Metaphor	Explanation
Wrist camera	Portrait studio	The software has the capability for transferring images from one device (e.g., PDA, PC) to another, and some image processing capabilities. It is much like a portrait studio, where a camera takes a picture, which is developed, retouched, printed, and distributed.
Wrist camera	Cities and Towns	(same assignment as above). Larger, more capable devices are like cities, in which many services are available. Smaller, less capable devices are like small cities, or even villages, where fewer services are available. Transfer of files is like a train moving from one municipality to another.
Variable transparency window	Chameleon	The window will use nanotechnology to vary opaqueness depending on sensor reading, e.g. temperature. It can also generate decorative patterns. This is the architecture project.

Refactoring

XP

Source: Fowler

"Disciplined technique for restructuring an existing body of code, altering its internal structure without changing its external behavior"

Example refactoring techniques:

- Encapsulate attribute (field)into function
- Replace conditional with dynamic binding
- Extract routine (method)
- Rename routine or attribute
- Move routine or attribute to another class
- Pull up, pull down

Used in agile methods as a substitute for upfront design

Agile Software Development

Bertrand Meyer

Part D: Practices
2: Development

What we have seen:

Practices that have exerted significant influence on the way we develop software: pair programming, refactoring

.. They are complements, not replacements, for more upfront-style techniques

ETH zürich

Agile software development

Bertrand Meyer

Part D: agile practices

1: Meetings

2: Development

3: Release

4: Testing and quality

5: Management and others

Only one pair integrates code at a time

Collective code ownership
Development proceeds in parallel

But: to avoid conflicts, only one pair is permitted to integrate its changes at any given time

Continuous integration

The combination of frequent releases with relentless testing

Keep system fully integrated at all times

Continuous integration

XP

Source: Wake

Rather than weekly or daily builds, build system several times per day

Benefits:

- Integration is easier because little has changed
- Team learns more quickly
- Unexpected interactions rooted out early: conflicts are found while team can still change approach
- Problematic code more likely to be fixed because more eyes see it sooner
- Duplication easier to eliminate because visible sooner

Release early and often

Follows from rejection of “Big Upfront Design”

Avoid long architectural phases

Refactor

Small releases

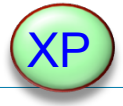
XP

Source: Jeffries

XP teams practice small releases in two important ways:

- Release running, tested software, delivering business value chosen by the Customer, every iteration. The Customer can use this software for any purpose, whether evaluation or even release to end users.
- Release to end users frequently as well. Web projects release as often as daily, in house projects monthly or more frequently. Even shrink-wrapped products are shipped as often as quarterly.

Incremental deployment



Deploy functionality gradually

“Big Bang” deployment is risky

Daily deployment

Goes back to Microsoft's Daily Build
"China Shop rules": you break it, you fix it

Difficult to reconcile with other XP principles

Ten-minute build

XP

Source: Shore 08

"Make sure that the build can be completed, through an automatic script, in ten minutes or less, to allow frequent integration. Includes:

- Compile source code
- Run tests
- Configure registry settings
- Initialize database schemas
- Set up web servers
- Launch processes
- Build installers
- Deploy

Make sure the build provides a clear indication of success or failure"

"If it has to take more than ten minutes, split the project into subprojects, and replace end-to-end funcational tests by unit tests"

Weekly cycle

XP

Source: Beck

Plan work a week at a time. Have a meeting at the beginning of every week:

- 1. Review progress, including how actual progress for the previous week matched expected progress
- 2. Have customers pick a week's worth of stories to implement this week
- 3. Break the stories into tasks

Start week by writing automated tests that will run when the stories are completed. Spend rest completing stories and getting tests to pass. The goal is to have deployable software at the end of the week.

The nice thing about a week is that everyone is focused on having the tests run on Friday. If you get to Wednesday and it is clear that all the tests won't be running, you still have time to choose the most valuable stories and complete them.

Quarterly cycle

Recommendation: reviews of high level system structure, goals and priorities on a quarterly basis, matching the financial reporting practices of many companies

Also an opportunity to reflect on the team practices and state of mind, and discuss any major changes in practices and tools

Period chosen as large enough not to interfere with current concerns, and short enough to allow frequent questioning of practices and updates of long-term goals

Agile Software Development

Bertrand Meyer

Part D: Practices
3: Release

What we have seen:

Agile methods promote frequent release cycles and continuous integration

Many variants (weekly, quarterly and in-between)

ETH zürich

Agile software development

Bertrand Meyer

Part D: agile practices

1: Meetings

2: Development

3: Release

4: Testing and quality

5: Management and others

Coding standards

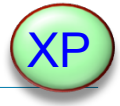
Project members all code to the same conventions

All code must have unit tests

Core idea of XP:

- Do not write code without associated unit tests

All code must pass unit tests before moving on



Code that does not pass tests is waste

Do not proceed to next step, e.g.

- Next user story
- Next release

until all tests pass

Test-First Development

Write tests **before** code

The test replaces the specification

Code the unit test first

XP

Source: Ron Jeffries

“Here is a really good way to develop new functionality:

- 1. Find out what you have to do.
- 2. Write a UnitTest for the desired new capability. Pick the smallest increment of new capability you can think of.
- 3. Run the UnitTest. If it succeeds, you're done; go to step 1, or if you are completely finished, go home.
- 4. Fix the immediate problem: maybe it's the fact that you didn't write the new method yet. Maybe the method doesn't quite work. Fix whatever it is. Go to step 3.

A key aspect of this process: don't try to implement two things at a time, don't try to fix two things at a time. Just do one.
When you get this right, development turns into a very pleasant cycle of testing, seeing a simple thing to fix, fixing it, testing, getting positive feedback all the way. Guaranteed flow. And you go so fast!

Try it, you'll like it.”

Test-Driven Development

Standard cycle:

- Add a test
- Run all tests and see if the new one fails
- Write some code
- Run the automated tests and see them succeed
- Refactor code

Expected benefits:

- Catch bugs early
- Write more tests
- Drive the design of the program
- Replace specifications by tests
- Use debugger less
- More modular code
- Better coverage
- Improve overall productivity

Test-Driven Development

The basic idea is sound…

… but not the replacement of specifications by test

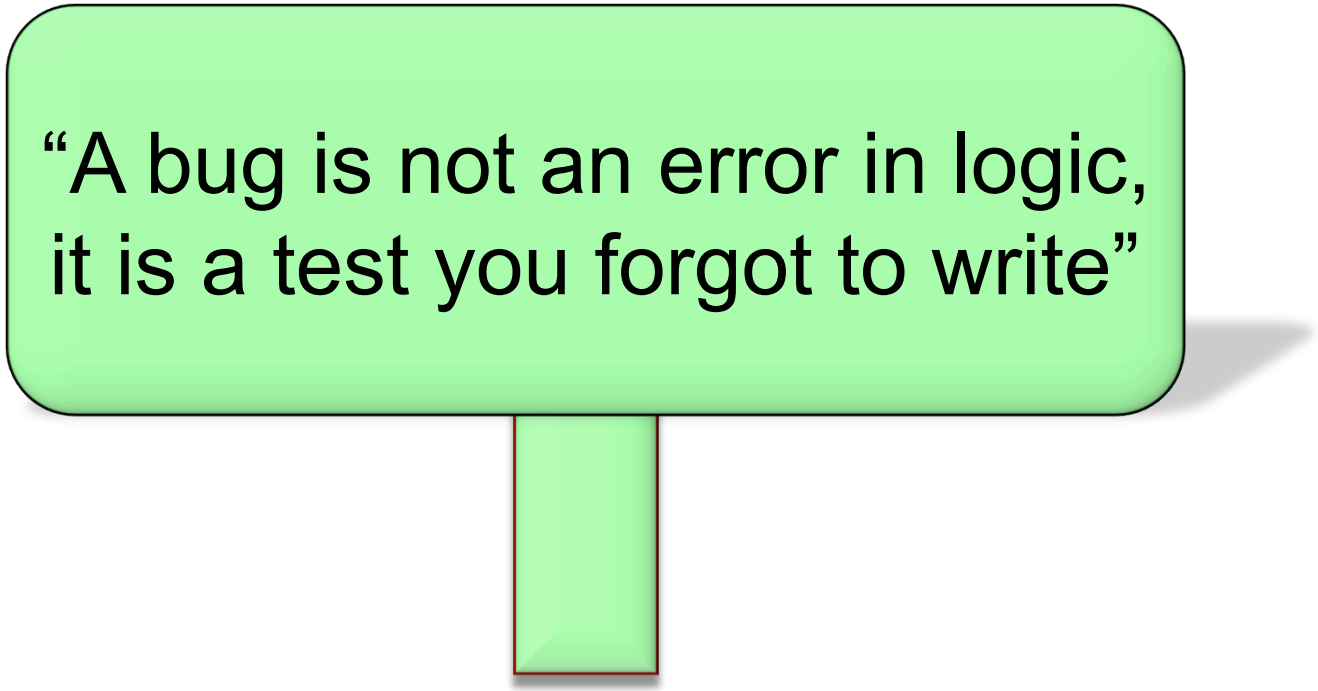
Major benefit: keep up-to-date collection of regression tests

Requirement that all tests pass can be unrealistic (tests degrade, a non-passing test can be a problem with the test and not with the software)

Basic TDD idea can be applied with specifications! See Contract-Driven Development

When bug found, create test before fixing it

XP



Root-cause analysis

When finding a defect, do not just fix it but analyze cause and make sure to correct that cause, not just the symptom

Run acceptance tests often and publish results

XP

Source: Wells

Acceptance tests are black box system tests. Each acceptance test represents some expected result from the system

Automate them so they can be run often

Publish acceptance test score is to the team

Agile Software Development

Bertrand Meyer

Part D: Practices
3: Testing and quality

What we have seen:

Tests drive development

Tests should all pass

Tests replace specifications

ETH zürich

Agile software development

Bertrand Meyer

Part D: agile practices

1: Meetings

2: Development

3: Release

4: Testing and quality

5: Management and others

Scrum of scrums

Each day after daily scrum

Clusters of teams discuss areas of overlap and integration

A designated person from each team attends

Agenda as Daily Scrum, plus the following four questions:

- What has your team done since we last met?
- What will your team do before we meet again?
- Is anything slowing your team down?
- Are you about to put something in another team's way?

Whole team

XP Scrum

Source: Jeffries (abridged)

All contributors sit together as members of one team:

- Includes a business representative who provides requirements, sets priorities and steers the project.
- Includes programmers
- May include testers
- May include analysts, helping to define requirements
- Often includes a coach
- May include a manager

None of these roles is the exclusive property of just one individual. The best teams have no specialists, only general contributors with special skills

Planning poker

Scrum

- Present individual stories for estimation
- Discuss
- Deck has successive numbers (quasi-Fibonacci)
- Each participant chooses estimate from his deck
- Keep estimates private until everyone has chosen a card
- Reveal estimates
- Repeat until consensus

(Variant of Wideband Delphi technique.)

Open workspace

Workspace:

- Organized around pairing stations
- With whiteboard space
- Locating people according to conversations they should overhear
- With room for personal effects
- With a place for private conversations

Expected benefits: improve communication, resolve problems quickly with the benefits of face-to-face interaction (as opposed to e.g. email)

Osmotic communication

Team is together in a room and listen to each other

Information to flow around it

Developer must break concentration

Information flows quickly throughout the team

Questions answered rapidly

All team updated on what is happening

Reduce need for email and other non-direct communication

Facilitate taking over of others' tasks

Informative workspace

Facilitate communication through well-organized workspace:

- Story board with user story cards movable from *not started* to *in progress* to *done* column
- Release charts
- Iteration burndown charts
- Automated indicators showing the status of the latest unit-testing run
- Meeting room with visible charts, whiteboards and flipcharts

Technical environment

Access to automated tests, configuration management, frequent integration, code repository

Team continuity

Keep the team together and stable

Do not reassign people to other teams or treat them as mere resources

Shrinking teams

XP

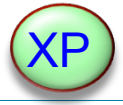
Source: Beck 05

As a team grows in capability, keep its workload constant but gradually reduce its size

This frees people to form more teams

When the team has too few members, merge it with another too-small team

Code and tests



Maintain only code and tests as permanent artifacts

Customer always available

XP

Source: Wells

All project phases require communication with customer, preferably face to face. Assign one or more customers to the development team who:

- Help write user stories to allow time estimates & assign priority
- Help make sure most of the desired functionality is covered by stories
- During planning meeting, negotiate selection of user stories for release
- Negotiate release timing
- Make decisions that affect their business goals
- Try system early to provide feedback
- (etc.)

"This may seem like a lot of the customer's time but the customer's time is saved initially by not requiring a detailed requirements specification and later by not delivering an uncooperative system"

Code and tests

Maintain only code and tests as permanent artifacts

Slack

XP

Source: Beck, deMarco

"In any plan, include some minor tasks that can be dropped if you get behind."

Goals:

- Establish trust in the team's ability to deliver
- Reduce waste

Agile Software Development

Bertrand Meyer

Part D: Practices
5: Management and others

What we have seen:

A number of management practices supporting the agile principles, in particular by ensuring that management gives developers the support and comfort they need