Manifesto for Agile Software Development

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

Kent Beck	James Grenning	Robert C. Martin
Mike Beedle	Jim Highsmith	Steve Mellor
Arie van Bennekum	Andrew Hunt	Ken Schwaber
Alistair Cockburn	Ron Jeffries	Jeff Sutherland
Ward Cunningham	Jon Kern	Dave Thomas
Martin Fowler	Brian Marick	

https://agilemanifesto.org/

Principles behind the Agile Manifesto

- *"Our highest priority is to satisfy the customer through early and continuous delivery of valuable software."*
- *"Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage."*
- *"Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale."*
- *"Business people and developers must work together daily throughout the project."*
- *"Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done."*
- *"The most efficient and effective method of conveying information to and within a development team is face-to-face conversation."*

Source: https://agilemanifesto.org/principles.html

Principles behind the Agile Manifesto

- *“Working software is the primary measure of progress.”*
- *“Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.”*
- *“Continuous attention to technical excellence and good design enhances agility.”*
- *“Simplicity – the art of maximizing the amount of work not done--is essential.”*
- *“The best architectures, requirements, and designs emerge from self-organizing teams.”*
- *“At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.”*

Source: https://agilemanifesto.org/principles.html

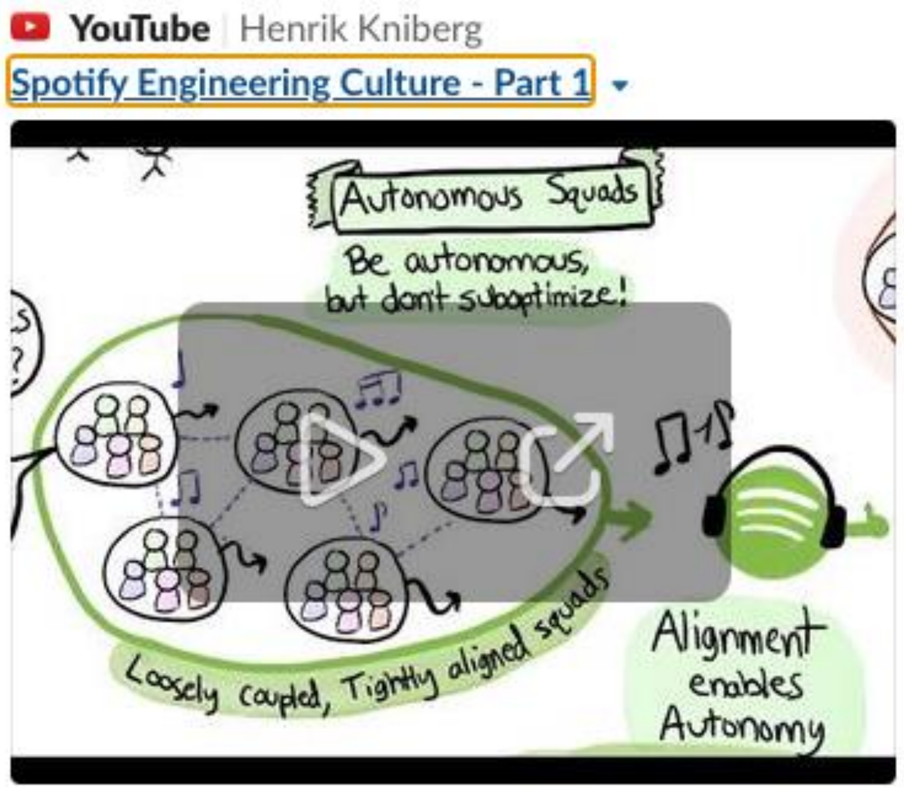
What is Agile?

- Agile is a set of values and principles that guide and shape development.
- There are a number of agile development methods that embody these values and principles in their practices:
  - Extreme Programming (XP)
  - Scrum
  - Kanban
  - Crystal Agile Framework
  - Dynamic System Development Method (DSDM)
  - Feature-Driven Development (FDD)

What is Agile?

- Agile is a set of values and principles that guide and shape development.
- There are a number of agile development methods that embody these values and principles in their practices:
  - Extreme Programming (XP)
  - Scrum
  - Kanban
  - Crystal Agile Framework
  - Dynamic System Development Method (DSDM)
  - Feature-Driven Development (FDD)

Case Study #1: Spotify


YouTube | Henrik Kniberg
Spotify Engineering Culture - Part 1
Autonomous Squads
Be autonomous, but don't suboptimize!
Loosely coupled, Tightly aligned squads
Alignment enables Autonomy

YouTube | Henrik Kniberg
Spotify Engineering Culture - Part 2
You are the culture
Model the behavior you want to see

https://www.youtube.com/watch?v=Yvfz4HGtoPc

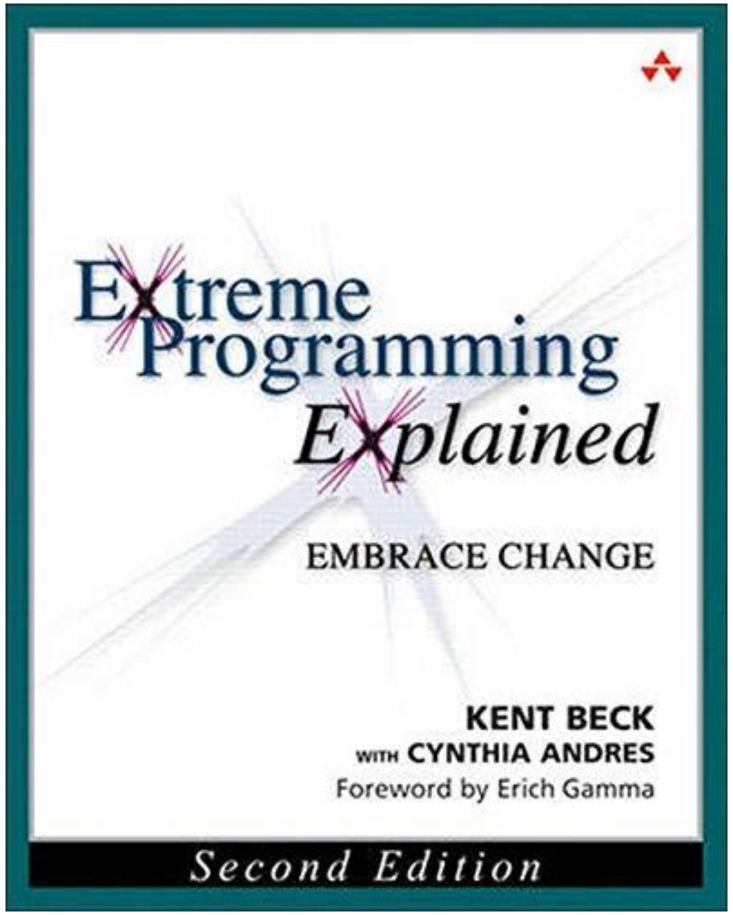
https://www.youtube.com/watch?v=vOt4BbWLWQw

What is Agile?

- Agile is a set of values and principles that guide and shape development.
- There are a number of agile development methods that embody these values and principles in their practices:
  - Extreme Programming (XP)
  - Scrum
  - Kanban
  - Crystal Agile Framework
  - Dynamic System Development Method (DSDM)
  - Feature-Driven Development (FDD)

What is Agile?

- Agile is a set of values and principles that guide and shape development.
- There are a number of agile development methods that embody these values and principles in their practices:
  - Extreme Programming (XP)
  - Scrum
  - Kanban
  - Crystal Agile Framework
  - Dynamic System Development Method (DSDM)
  - Feature-Driven Development (FDD)



Note to Programmers

"Even programmers can be whole people in the real world. XP is an opportunity to test yourself, to be yourself, to realize that maybe you've been fine all along and just hanging with the wrong crowd."

- Above note from Kent Beck with Cynthia Andres, Extreme Programming Explained (2/E)

Extreme Programming

- Extreme Programming has
  - Values,
  - Principles and
  - Practices

Values of XP

- Communication, Simplicity, Feedback, Courage, Respect…

“Values bring purpose to practice”

Extreme Programming

Principles of XP

- Humanity - software is developed by people
- Economics - software costs money
- Mutual Benefit - software activities should benefit everybody
- Self-Similarity - try reusing solutions across projects
- Improvement - perfect software doesn’t exist
- Diversity - different skills benefit a software team
- Quality - can not be sacrificed
- Other Principles: Flow, Opportunity, Redundancy, Failure, Baby Steps, Accepted Responsibility, Reflection

Extreme Programming

A Modern, Lightweight, Lean Software Process

- Originally used for small- to medium-sized software projects (Although it can scale to larger projects)
- Designed to adapt well to the observed realities of modern software production:
  - short timelines
  - high expectations
  - severe competition
  - unclear and rapidly changing requirements
- Based on the idea of continuous evolution
- Very practical – based largely on simplicity, testing

What’s So Extreme About It?

Why is it called Extreme?

- When first conceived, the idea was to take the best practices of good software development to the limit (the extreme)
  - if code reviews are good, review code all the time
  - if testing is good, test all the time
  - if design is important, design all the time
  - if simplicity is good, always use the simplest solution possible
  - if architecture is important, refine architecture all the time
  - if integration is important, integrate all the time
  - if short iterations are good, use shortest iterations possible

Oh No! Not Yet Another Process ...

Why Make a Different Approach?

- XP was born of the dissatisfaction of programmers with the actual situation in most software development environments
- Frustration the lack of time to test adequately because of the rush to get new software and new versions out quickly
- Dissatisfaction with the lack of ongoing advice and social support for difficult technical decisions, and management blame for decisions that do not turn out well
- Worry about lack of connection between planning and design activities and actual source code
- Worry about the communication gap between management and technical staff

Properties of Extreme Programming

Characteristics of XP

- continuing feedback from short cycles
- incremental planning that evolves with the project
- responsive flexibility in scheduling
- heavy and continuous use of testing and test automation
- emphasis on close and continuous collaboration and communication
- use of tests and source code as primary communication media (communication at programmer's level)
- evolutionary model from conception to retirement of system
- emphasis on small, short term practices that help yield high quality long-term results

Attacking Risks Before They Arise

(1) Schedule Slips

- Software isn’t ready on the scheduled delivery date
- Addressed in XP by short release cycles, frequent delivery of intermediate versions to customers, customer involvement and feedback in development of software

(2) Project Cancellation

- After several schedule slips, the project is cancelled
- Addressed in XP by making the smallest initial release that can work, and putting it into production early – thus establishing credibility and results

Attacking Risks Before They Arise

(3) System Defect Too High, or Degrades with Maintenance

- Software put in production, but defect rate is too high, or after a year or two of changes rises so quickly, that system must be discarded or replaced
- Addressed in XP by creating and maintaining a comprehensive set of tests run and re-run after every change, so defect rate cannot rise
- Programmers maintain tests function-by-function, users maintain tests system feature-by-system feature

(4) Business Misunderstood

- Software put in production, but doesn’t solve the problem it was supposed to
- Addressed in XP by making customer an integral part of the team, so team is continually refining specification to meet expectations

Attacking Risks Before They Arise

(5) Business Changes

- Software put in production, but business problem it is designed for changes or is superseded by new, more pressing business problems
- Addressed in XP using short release cycles and by having customer as an integral part of the team
- Customer helps team continually refine specification as business issues change, adapting to new problems as they arise -programmers don’t even notice

(6) Featuritis (or False Feature Risk)

- Software has a lot of neat-o potentially interesting features, which were fun to implement, but don’t help customer make more money
- Addressed in XP by addressing only the highest priority tasks, maintaining focus on real problems to solve

Attacking Risks Before They Arise

(7) Staff Turnover

- After a while, the best programmers begin to hate the same old program, get bored and leave
- In XP programmers make their own estimates and schedules, get to plan their own time and effort, get to test thoroughly
- Less likely to get frustrated with impossible schedules and expectations
- In XP emphasis is on day to day social human interaction, pair and team effort and decisions
- Less likely to feel isolated and unsupported

XP in Practice

Practices of XP

- We will now look at the actual practices of the XP process.
- In XP, primary practices are good practices to start with when beginning with XP (we will focus mainly on these in our project)
- In XP, corollary practices are for experience XP teams. These practices are dangerous without first mastering the primary practices.

XP in Practice - Planning Practices

Stories

- Story – “a unit of customer-visible functionality”
- Each story should have
  - a name,
  - a short description (written or graphical), and
  - an estimate of the implementation effort required.
- Usually written on index cards and placed on a wall in the office.

In our project, the requirements document can be viewed as a collection of stories

XP in Practice - Planning Practices

Cycles

- Weekly Cycle – Plan one week at a time.
  - Have a weekly meeting to
    - discuss last week's actual vs. expected progress
    - pick stories to implement this week. Each story is broken into tasks (effort for each task is estimated).
- Quarterly Cycle – Plan one quarter at a time
  - Have a quarterly planning meeting to
    - reflect on the team and project with respect to large goals.
    - plan themes for the quarter and pick stories for each theme.

In our project, we don't use these practices

XP in Practice - Planning Practices

Slack

- Plan for slack and build slack into plan
- Don’t underestimate the effort to implement stories.

In our project, we don’t use these practices

XP in Practice - Planning Practices

The Planning Game - Business vs. Technical Constraints

- The Planning Game refers to the practice of having a continuous dialog between business and technical people on the project
- In weekly meetings, business people bring business constraints, and technical people bring technical constraints
- Business people bring issues of scope, priority, releases
- Technical people bring estimates, consequences, scheduling
- Forces the project members to continually balance between what is possible (the technical aspects) and what is desirable (the business aspects)

In our project, we don't use these practices

XP in Practice - Planning Practices

Plan for Small Releases

- Small Releases refers to the practice of addressing only the most pressing business requirements, and getting them addressed by releasing a new version quickly
- Means that we should bring the first version into production as quickly as possible
- Means that we should shrink the cycle to the next version as much as possible

In our project we use quick releases at roughly two week intervals (the length of project phase submissions)

XP in Practice – Programming Practices

Pair Programming

- Pair Programming refers to the practice of having all production code written with two people working together on one terminal
- One partner works tactically, on the specific part of the code (e.g. method) being coded at the moment
- The other partner works strategically, considering higher level issues such as:
  - is this approach going to work?
  - can we simplify this by restructuring?
  - what other tests do we need to address here?

In our project, we can do programming in pairs

XP in Practice – Programming Practices

Test First/Test Driven Programming

- The only required program features are those for which there is an automated test
- Always create tests first, and treat them as the goal (specification)
- Programmers create unit tests (tests for each method or segment of code)
- Customers create functional (acceptance) tests (tests that check that the product has the required functionality)

In our project, we create explicit tests first as we go along, and program to meet our tests

XP in Practice – Programming Practices

Incremental Design

- Improving and work on the design of the system every day
- Refactoring is part of incremental design and refers to the practice of continually looking for ways to simplify the architecture and coding of the system as new features and changes are made
- When a new feature or change is needed, we first look to see if there is a way to rearchitect the system to make it easier or simpler to add – if so, we rearchitect first
- Once the new feature has been added or changed, we look to see if the resulting new program can be simplified by rearchitecting or merging similar code

In our project, we face changes that may require incremental design and refactoring through clarifications from the client

XP in Practice – Programming Practices

Coding Standards

- Coding Standards are project-wide conventions about the coding of programs
- Necessary since everyone is responsible for all of the code, and may have to read or change any part of it at any time
- Usually specifies:
  - commenting standards, e.g., every method must have a comment of the form …
  - naming conventions, e.g., variables representing dates will always be named ending in "Date", all constants will be named with a two-letter prefix indicating their business type, etc.

In our project, you are required to specify our coding standards, and they will be judged according to the clarity, readability and consistency of your code

XP in Practice - Integration Practices

Continuous Integration

- In XP, new code is always integrated and tested within a day
- Changes are not allowed to go on without being continually tested in context, to catch integration failures before they happen

In our project, we can model this by testing again immediately after each day's changes

Ten-Minute Build

- Automatically building the entire system and running all the tests should take no more than 10 minutes
- A short build means more chances for feedback.

In our project, building the system and running all the test cases should ideally not take more then 10 minutes

XP in Practice - General Practices

The following general practices are primarily related to the environment of an XP team.

Sit Together

- The whole team should work in an open space.

Whole Team

- The whole team means having people with the necessary skills and the right attitude.

Informative Workspace

- Make the workspace about work (e.g., visual display of project information such as stories).

In our project and under normal circumstances we we should strive to have the right XP environment but this is not possible during the pandemic.

XP in Practice – Corollary Practices

Corollary Practices by Category

- Business practices: negotiated scope contract, pay-per-use, daily deployment.
- Programming practices: single code base, shared code, code & test.
- Team practices: team continuity, shrinking teams, real customer involvement, root cause analysis.

In our project, we will not use corollary practices

Agile Methods I

Summary

- In general, agile methods have become a dominant software development approach
- Extreme Programming is one example of a lean and programmer-centred agile method. It uses a set of standard practices that together form an easy to apply system for team development of software

References

- https://agilemanifesto.org/
- *Extreme Programming Explained (2/E) by* Beck – Chapters 6, 7, 9
- http://www.extremeprogramming.org/