Context

1. The agile manifesto

the 15-year-old text which first presented Agile ideas (2001)
agilemanifesto.org

12 principles

- 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 software frequently from a couple of weeks to a couple of months, with a preference to the shorter time scale.
- businesspeople and developers will work together daily throughout a project
- build projects around individuals. Give them the tools, give them the support that they need and trust them to get the job done.
- the most efficient and effective method of conveying information to and within the development team is face to face conversation.
- Working software is the primary measure of progress.
- promote sustainable development, make sure that everyone can maintain a constant pace indefinitely.
- technical excellence.
- Simplicity-the art of maximizing the amount of work not done.
- the best architecture's requirements and designs emerge from self-organizing teams.
- at regular intervals, meet and discuss what you have done in order to reflect on the successes and failures, and in order of course to adjust the team's behavior accordingly.

2. Agile methods
   - ***XP (extreme programming, Kent Beck),***

Background: 90s, popularity of UML, CMMI(Capability Maturity Model Integration), against this, what counts is the program

- ***Lean software (Mary Poppendieck)***

  Apply to programming successful principles and methods in other engineering fields

Ex: Toyota attention to get rid of waste

- ***Crystal (Alistair Cockburn)***

Try to combine agile with traditional process management

- ***Scrum (Ken Schwaber and Ken Sutherland)***

Importance of self-organized teams, more of management rather than technical, importance of short release iterations—Sprints
Negotiated scope contract: reduce risk by signing a sequence of short contracts instead of one long

3. Official agile principles
   Issues: many practices, assertions, no testing

4. Agile values
   New reduced role for manager—don't assign tasks to project members
   No 'Big Upfront' steps—start coding right away
   Iterative development
   Limited, negotiated scope—only build functionalities that are strictly needed
   Focus on quality, achieved through testing

Principle

1. The enemy: Big Upfront Anything
   - ***Organizational***

Ø 1 Put the customer at the center
Ø 2 Accept change
Ø 3 Let the team self-organize
Ø 4 Maintain a sustainable pace
Ø 5 Produce minimal software:
• 5.1 Produce minimal functionality
• 5.2 Produce only the product requested
• 5.3 Develop only code and tests

- ***Technical***

Ø 6 Develop iteratively
• 6.1 Produce frequent working iterations
• 6.2 Freeze requirements during iterations
Ø 7 Treat tests as a key resource:
• 7.1 Do not start any new development until all tests pass
• 7.2 Test first
Ø 8 Express requirements through scenarios

- ***Lifecycle models (Origin: Royce, 1970, Waterfall model)***

describe the set of processes involved in the production of software systems, and their sequencing
system requirement—software requirement—analysis—program design—coding—testing—operations
feasibility study—requirement—specification—global design—detailed design—implementation—V&V--distribution

- problem

§ Late appearance of actual code
§ Lack of support for requirements change — and more generally for extendibility and reusability
§ Lack of support for the maintenance activity (70% of software costs?)
§ Division of labor hampering Total Quality Management
§ Impedance mismatches
§ Highly synchronous model

- The two agile criticisms of requirements

Change criticism
Waste criticism

2. Organizational principles
   - ***Put the customer at the center***

XP: embedded customer; Scrum: product owner
Can customer involvement replace requirements?

   - ***Accept change***

In standard software engineering, especially object-oriented: extendibility

   - ***Let the team self-organize***

Agile view: managers listen to developers, explain possible actions, provide suggestions for improvements.
"The leader is there to:
Encourage progress
Help catch errors
Remove impediments
Provide support and help in difficult situations
Make sure that skepticism does not ruin the team's spirit"
Team chooses own commitments & has access to customers

   - ***Maintain a sustainable pace***

Frequent code-merge
Always maintain executable, test-covered, high-quality code
Constant refactoring, helping keep fresh and alert minds
Collaborative style
Constant testing

3. More organizational principles
   - ***Minimalism:***

Ø Minimal functionality-- work on the story we have
The "lean" view: Seven wastes of software development:
Extra/Unused features (Overproduction)
Partially developed work not released (Inventory)
Intermediate/unused artifacts (Extra Processing)
Seeking Information (Motion)
Escaped defects not caught by tests/reviews (Defects)
Waiting (including Customer Waiting)
Handoffs (Transportation)

   - ***Product only***

What's the simplest thing that could possibly work?
reuse

- ***Only code and tests***

You get no credit for any item that does not result in running, tested code

4. Technical principles
    - ***Iterativeness:***

Ø Frequent working iterations
Database-networking-business logic-user interface, horizontally and vertically
Ø Freeze requirements during iteration (Closed-Window Rule: During an iteration, no one may add functionality)
Ø Dual development:
Early on: build infrastructure
Later: product release

- ***Tests: do not move on until all tests pass***

Need to classify severity
Test first

- ***Express requirements through scenarios***

User stories
"As a <user_or_role>
I want <business_functionality>
so that <business_justification>"
--Example:
"As a customer,
I want to see a list of my recent orders,
so that I can track my purchases with a company."

Major application: for ***validating*** requirements
BUT!! They cannot define the requirements:
Ø Not abstract enough
Ø Too specific
Ø Describe current processes
Ø Do not support evolution

5. A few method-specific principles
    - ***Lean***

Eliminate waste
Ø Unnecessary code
Ø Unnecessary functionality
Ø Delay in process
Ø Unclear requirements
Ø Insufficient testing
Ø Avoidable process repetition
Ø Bureaucracy
Ø Slow internal communication
Ø Partially done coding
Ø Waiting for other activities, team, processes

Ø Defects, lower quality
Ø Managerial overhead
Value stream mapping: strategy to recognize waste.
Amplify learning
Ø To prevent accumulation of defects, run tests as soon as the code is written
Ø Instead of adding documentation or planning, try different ideas by writing and testing code and building
Ø Present screens to end-users and get their input
Ø Enforce short iteration cycles, each including refactoring and integration testing
Ø Set up feedback sessions with customers
Decide as late as possible
until they can be made based on facts, not assumptions, and customers better understand their needs
Multiple design
At period end, the surviving designs are compared and one chosen, perhaps with modifications based on learning from the others

- ***Crystal***

Focus
Focus on individual task, to ensure progress:
Ø Control flow of progress
Ø Deal with interruptions:
• Two-hour period without interruption
• Assign developer to project for at least two days before switching
Focus on direction of project
Ø Define goals clearly
Ø Prioritize goals
Personal safety
Encourage free expression of ideas

- ***Scrum***

Minimizing dependencies
it is possible to remove dependencies between user stories, so that at any point any user story can be selected according to the proper criteria (maximizing business value)

- ***XP***

Humanity
Offer developers what they expect:
Ø Safety
Ø Accomplishment
Ø Belonging
Ø Growth
Ø Intimacy

- ***Lean & Scrum***

Deliver as fast as possible

It is not the biggest that survives, but the fastest

Roles

1. Traditional (non-agile) manager responsibilities

Define goals
Define deadlines
Assign tasks
Provide interface with higher management
Provide interface with customer
Validate requirements
Decide whether goals have been met
Enforce deadlines
Coach, mentor
Enforce rules and methodology

2. Three Scrum roles

In Scrum there is no "project manager" or "team leader"
The preceding tasks are split between:
Ø (Self-organizing) team
Ø Product owner
Ø Scrum Master

- ***Team***

The team:
Ø Is cross-functional
Ø Has 7+/- 2 members
Ø Selects iteration goal and work results
Ø Organizes itself and its work
Ø Can do everything within guidelines to reach goal
Ø Demos work results to product owner

Core participants and fellow travelers (Scrum)

Core participants are truly "committed" to the project
Fellow travelers are "involved". They should stand on the side in discussions, giving their opinion if invited to do so
"Pigs" and "chickens"

- ***Product owner***

Ø Defines product features
Ø Decides on release date
Ø Decides on release content
Ø Responsible for product profitability (ROI)
Ø Prioritizes features according to market value
Ø Can change features and priority over 30 days
Ø Accepts or rejects work results

- ***Scrum Master***

Ø Ensures that the team is functional and productive

Ø Enables cooperation across all roles & functions
Ø Shields team from external interferences
Ø Enforces process: daily meeting, planning & review meetings
Ø Removes impediments
Hardware limitations
Missing requirements (recall "waste" in Lean)
Missing supporting software (from within the team, or
outside of it)
Management interference
Bureaucratic delays
Ø Normally, does no develop

3. Other agile roles
    - Expert user

Person with expert knowledge of the project area, who can answer questions and suggest solutions to problems
Minimum of once a week, two-hour meeting with expert user, and ability to make phone calls

- Customer

Customer responsibilities in XP:
Ø Trust developers' technical decisions, because developers understand technology
Ø Analyze risk correctly, weighing stories against each other
Ø Provide precise stories, enabling developers to produce comprehensive task cards and accurate estimates
Ø Choose stories with maximum value, scheduling the most valuable stories that could possibly fit in to next iteration
Ø Work within team, providing guidance and receiving feedback as quickly and accurately as possible

- Developer

Main job: turn customer stories into working code.
Avoid need to make business decisions, by allowing the customer to make them

- Tracker

Keeps track of the schedule
Most important metric
Ø Velocity: ratio of ideal time estimated for tasks to actual time spent
implementing them
Other important data:
Ø Changes in velocity
Ø Amount of overtime worked
Ø Ratio of passing to failing tests
These numbers measure progress and the rate of progress and help determine if the project is on schedule for the iteration
To measure velocity within the iteration, every day or two, the tracker asks each developer how many tasks he has completed

- Coach

Optional role:
Ø Guides team

Ø Mentors team
Ø Leads by example
Ø Teaches when necessary
Ø May teach by doing
Ø May offer ideas to solve thorny problems
Ø May serve as intermediary with management

Practices

1. Meetings

Scrum

- *Daily meetings*

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*:

Ø Defining commitments
Ø Uncovering impediments
Resolution will take place outside of meeting

- *Three questions*

Ø What did you do yesterday?
Ø What will you do today?
Ø 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
Ø 1st half, product owner + team: prioritize product backlog
Ø 2nd half, team only: plan for Sprint, producing sprint backlog

- *Retrospective*

All team members reflect on past sprint
Make continuous process improvements
Two main questions:
Ø What went well?
Ø What could be improved?
3-hour time limit

- *Review meeting*

Review work:
Ø Completed
Ø Not completed
Present and demo completed work to stakeholders
Incomplete work cannot be demonstrated
4-hour time limit

Crystal

- *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

2. Development

XP:

- *Pair programming*

Two programmers sitting at one machine
Thinking out loud
Goals:
Ø Make thinking process explicit
Ø Keep each other on task
Ø Brainstorm refinements to system
Ø Clarify ideas
Ø Take initiative when other stuck, lowering frustration
Ø 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:
Ø Most non-trivial features extend across many layers in the application
Ø Code ownership creates unnecessary dependencies between team members and delays
Ø What counts is implemented features, not personal responsibility
Ø Avoid blame game
Ø Avoid specialization
Ø Minimize risk (team members leaving)

- *Leave optimization till last*

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

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

- *System metaphor*

"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"

- *Refactoring*

"Disciplined technique for restructuring an existing body of code, altering its internal structure without changing its external behavior"
Example refactoring techniques:
Ø Encapsulate attribute (field)into function
Ø Replace conditional with dynamic binding
Ø Extract routine (method)
Ø Rename routine or attribute
Ø Move routine or attribute to another class
Ø Pull up, pull down

3. Release

XP

- *Only one pair integrates code at a time*

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
build system several times per day
Benefits:
Ø Integration is easier because little has changed
Ø Team learns more quickly
Ø Unexpected interactions rooted out early: conflicts are found while team can still change approach
Ø Problematic code more likely to be fixed because more eyes see it sooner
Ø 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*

Ø 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.
Ø 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 (XP & Scrum)*

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*

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

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

- *Weekly cycle (ideal release cycle)*

Plan work a week at a time. Have a meeting at the beginning of every week:
Ø 1. Review progress, including how actual progress for the previous week matched expected progress
Ø 2. Have customers pick a week's worth of stories to implement this week
Ø 3. Break the stories into tasks

- *Quarterly cycle*

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

4. Testing and quality

XP

- *Coding standards*

Project members all code to the same conventions

- *All code must have unit tests*

Core idea of XP: ØDo 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.
Ø Next user story
Ø Next release
until all tests pass

- *Test-First Development*

Write tests before code. The test replaces the specification

- *Code the unit test first*

"Here is a really good way to develop new functionality:
Ø 1. Find out what you have to do.
Ø 2. Write a UnitTest for the desired new capability. Pick the smallest increment of new capability you can think of.
Ø 3. Run the UnitTest. If it succeeds, you're done; go to step 1, or if you are completely finished, go home.
Ø 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.

- *Test-Driven Development*

Standard cycle:
Ø Add a test
Ø Run all tests and see if the new one fails
Ø Write some code

Ø Run the automated tests and see them succeed
Ø Refactor code
Expected benefits:
Ø Catch bugs early
Ø Write more tests
Ø Drive the design of the program
Ø Replace specifications by tests
Ø Use debugger less
Ø More modular code
Ø Better coverage
Ø Improve overall productivity

- *When bug found, create test before fixing it*
- *Root-cause analysis (XP & Scrum)*

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*

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

5. Management and others

Scrum

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

- *Whole team*

All contributors sit together as members of one team:
Ø Includes a business representative who provides requirements, sets priorities and steers the project.
Ø Includes programmers
Ø May include testers
Ø May include analysts, helping to define requirements
Ø Often includes a coach
Ø May include a manager

- *Planning poker*

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

Ø Reveal estimates
Ø Repeat until consensus

XP & Crystal

- *Open workspace (Osmotic communication)*

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

XP & Scrum

- *Informative workspace*

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

Crystal

- *Technical environment*

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

XP

- *Team continuity*

Keep the team together and stable
Do not reassign people to other teams or treat them as mere resources

- *Shrinking teams*

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

- *Code and tests*

Maintain only code and tests as permanent artifacts

- *Customer always available*

Assign one or more customers to the development team who:
Ø Help write user stories to allow time estimates & assign priority
Ø Help make sure most of the desired functionality is covered by stories
Ø During planning meeting, negotiate selection of user stories for release
Ø Negotiate release timing
Ø Make decisions that affect their business goals
Ø Try system early to provide feedback

- *Slack*

“In any plan, include some minor tasks that can be dropped if you get behind.”
Goals:
Ø Establish trust in the team's ability to deliver
Ø Reduce waste

Artifacts

From user stories to burndown charts

- *User story(XP & Scrum)*

Defines an atom of functionality
Story card and task card (XP)

- *Use case*

Describes how to achieve single business goal or task through the interactions between external actors and system
One of the UML diagram types

- *Use case VS user story*

User story:
Ø Very simple
Ø Written by customer
Ø Incomplete, possibly inaccurate
Ø Does not handle exceptional cases
Ø Starting point for additional discussions with customer
Use case:
Ø More complex
Ø Written by developer in cooperation with customer
Ø Attempts to be complete, accurate
Ø Should handle all possible cases
Ø Intended to answer any developer questions about customer requirements without further interaction with customer

- *Product backlog (Scrum, Visualized in "task board")*

Ø Maintained throughout project
Ø Property of product owner
Ø Open and editable by anyone
Ø Contains backlog items: broad descriptions of all potential features, prioritized by business value
Ø Includes estimates of business value
Ø Includes estimates of development effort, set by team

- *Task board, story board (Scrum)*

Used to see and change the state of the tasks of the current sprint: "to do", "in progress", "done".

- *Velocity*

Measure of progress in a project: Number of items delivered
Ø Measured in tasks, user stories, backlog items…

- *Burndown chart*

Publicly displayed chart, updated every day, showing, for the sprint backlog:
Ø Remaining work
Ø Progress

- *Bullpen (XP)*

Single, open room

Assessments

- *The Ugly*

Ø Rejection of upfront tasks
Ø Particularly: no upfront requirements
Ø Dismissal of a priori architecture work
Ø User stories as a replacement for abstract requirements
Ø Tests as a replacement for specifications
Ø Feature-based development & ignorance of dependencies
Ø Method keeper (e.g. Scrum Master) as a separate role
Ø Test-driven development (but not the rest of agile's emphasis on tests)
Ø Dismissal of traditional manager tasks
Ø Dismissal of auxiliary products and non-shippable artifacts
Ø Dismissal of a priori concern for extendibility
Ø Dismissal of a priori concern for reusability

- *The indifferent*

ØPair programming
ØOpen-space working arrangements
ØSelf-organizing teams
ØMaintaining a sustainable pace
ØProducing minimal functionality
ØPlanning poker
ØCross-functional teams
ØEmbedded customer

- *The good*

Ø Acceptance of change
Ø Iterative development
Ø Emphasis on working code
Ø Tests as one of the key resources of the project
Ø Constant test regression analysis
Ø Notion of velocity
Ø No branching
Ø Product (but not user stories!) burndown chart
Ø Daily meeting

- *The brilliant*

Ø Short iterations
Ø Closed-window rule
Ø Refactoring (but not as a substitute for design)
Ø Associating a test with every piece of functionality
Ø Continuous integration