

Different approaches to agile software development

Inf2-SEPP

Adriana Sejfia

School of Informatics, University of Edinburgh

Last lecture

- Plan-driven processes (up to the late '90s):
 - The Waterfall Model
 - The Spiral Model

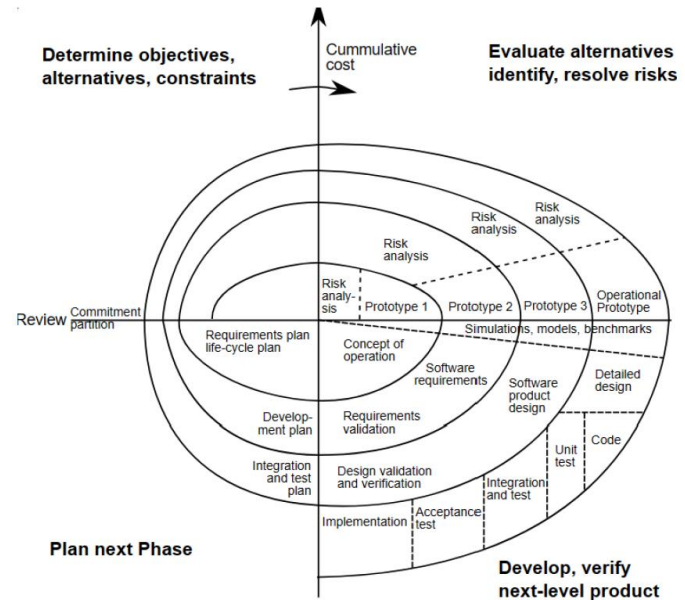
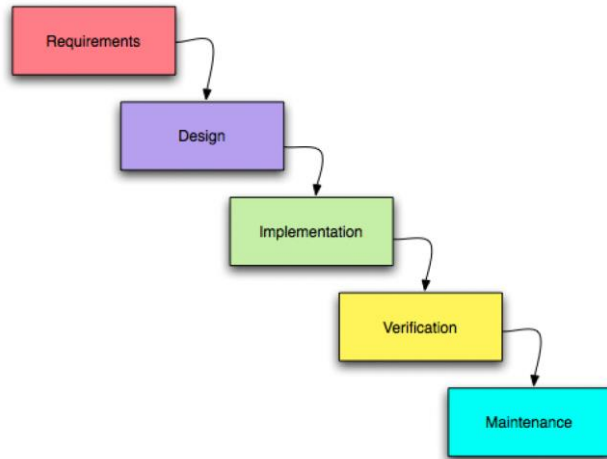
This lecture

- Different approaches to agile software development
 - Reminders
 - Extreme Programming (XP)
 - Definition and values
 - Example risks and XP responses
 - XP classification of software development activities
 - XP practices
 - Can one mix and match XP practices?
 - Applicability of XP
 - Scrum
 - Definition, values
 - The Scrum team
 - Scrum events
 - Applicability of Scrum

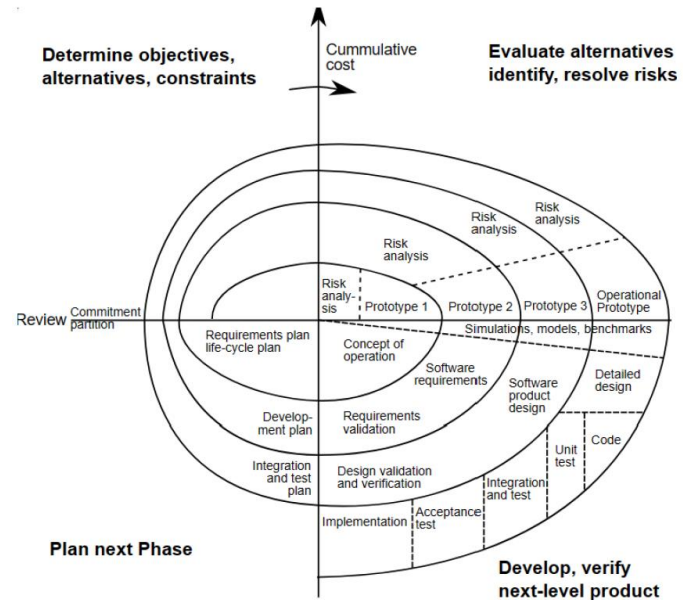
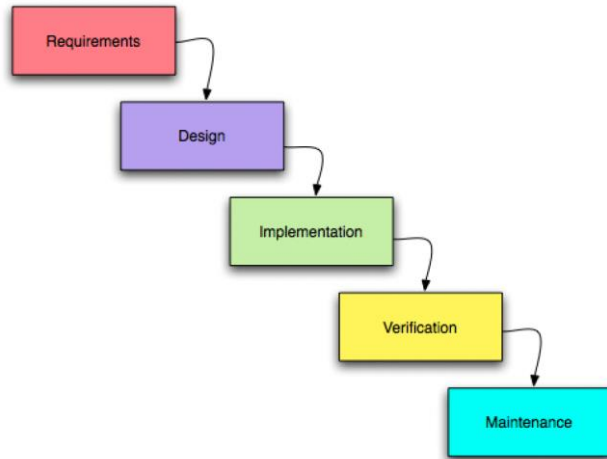
This lecture (continued)

- Kanban
 - Definition, theory
 - Kanban practices, including the Kanban board
 - Kanban measures
 - Applicability of Kanban

Reminder: Plan-driven Waterfall and Spiral models

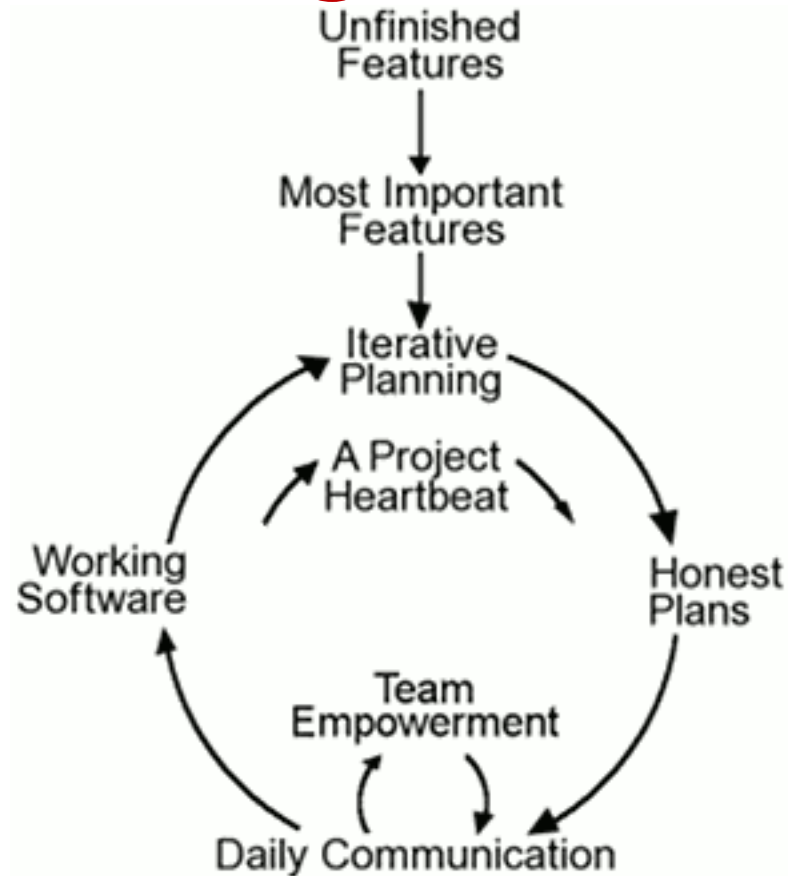


Reminder: Plan-driven Waterfall and Spiral models

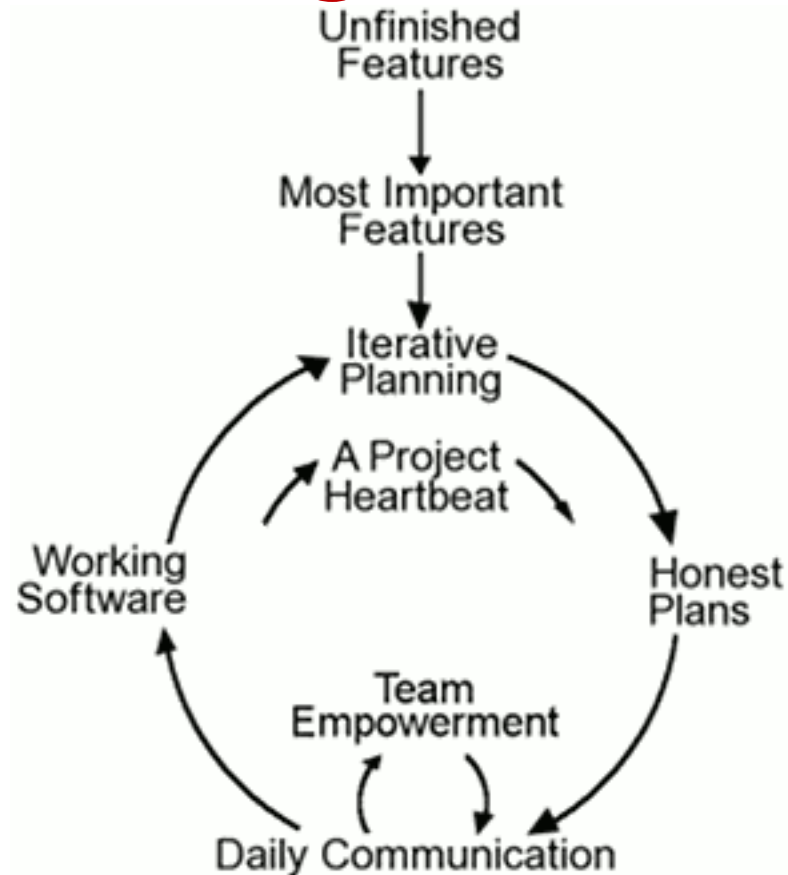


What the Spiral model was reaching towards was that software development has to be agile: able to react quickly to change.

Reminder: Agile



Reminder: Agile



IMPORTANT! By "customer" we will always mean initiator of requirements (even business person from development company!) in following slides.

Extreme Programming

- Extreme Programming (XP) is "a humanistic discipline of software development, based on values of communication, simplicity, feedback and courage"

Extreme Programming

- Extreme Programming (XP) is "a humanistic discipline of software development, based on values of communication, simplicity, feedback and courage"
- People: Kent Beck, Ward Cunningham, Ron Jeffries, Martin Fowler, Erich Gamma...

Extreme Programming

- Extreme Programming (XP) is "a humanistic discipline of software development, based on values of communication, simplicity, feedback and courage"
- People: Kent Beck, Ward Cunningham, Ron Jeffries, Martin Fowler, Erich Gamma...
- More info: www.extremeprogramming.org

Extreme Programming

- Extreme Programming (XP) is "a humanistic discipline of software development, based on values of communication, simplicity, feedback and courage"
- People: Kent Beck, Ward Cunningham, Ron Jeffries, Martin Fowler, Erich Gamma...
- More info: www.extremeprogramming.org
- Beck "Extreme Programming Explained: Embrace Change"

Example risks and the XP responses

- Schedule slips: Short iterations give frequent feedback; features prioritised

Example risks and the XP responses

- Schedule slips: Short iterations give frequent feedback; features prioritised
- Project cancelled after many slips: Customer chooses smallest release with biggest value

Example risks and the XP responses

- Schedule slips: Short iterations give frequent feedback; features prioritised
- Project cancelled after many slips: Customer chooses smallest release with biggest value
- Release has so many defects it is never used: Tests written with both unit-level and customer perspectives

Example risks and the XP responses

- Schedule slips: Short iterations give frequent feedback; features prioritised
- Project cancelled after many slips: Customer chooses smallest release with biggest value
- Release has so many defects it is never used: Tests written with both unit-level and customer perspectives
- System degrades after release: Frequent rerunning of tests maintains quality

Example risks and the XP responses

- Schedule slips: Short iterations give frequent feedback; features prioritised
- Project cancelled after many slips: Customer chooses smallest release with biggest value
- Release has so many defects it is never used: Tests written with both unit-level and customer perspectives
- System degrades after release: Frequent rerunning of tests maintains quality
- Business misunderstood: Customer representative embedded in development team

Example risks and the XP responses

- Schedule slips: Short iterations give frequent feedback; features prioritised
- Project cancelled after many slips: Customer chooses smallest release with biggest value
- Release has so many defects it is never used: Tests written with both unit-level and customer perspectives
- System degrades after release: Frequent rerunning of tests maintains quality
- Business misunderstood: Customer representative embedded in development team
- System rich in unimportant features: Only highest priority tasks addressed

Example risks and the XP responses

- Schedule slips: Short iterations give frequent feedback; features prioritised
- Project cancelled after many slips: Customer chooses smallest release with biggest value
- Release has so many defects it is never used: Tests written with both unit-level and customer perspectives
- System degrades after release: Frequent rerunning of tests maintains quality
- Business misunderstood: Customer representative embedded in development team
- System rich in unimportant features: Only highest priority tasks addressed
- Staff turnover: Programmers estimate task times; new programmers nurtured

XP classification of software development activities

- Coding
 - Central. Includes understanding, communicating, learning

XP classification of software development activities

- Coding
 - Central. Includes understanding, communicating, learning
- Testing
 - Embodying requirements, assessing quality, driving coding

XP classification of software development activities

- Coding
 - Central. Includes understanding, communicating, learning
- Testing
 - Embodying requirements, assessing quality, driving coding
- Listening
 - Understanding the customer, communicating efficiently

XP classification of software development activities

- Coding
 - Central. Includes understanding, communicating, learning
- Testing
 - Embodying requirements, assessing quality, driving coding
- Listening
 - Understanding the customer, communicating efficiently
- Designing
 - Creating structure, organising system logic

XP practices

- The Planning Game
- On-site customer
- Small releases
- Metaphor
- Continuous integration
- Simple design
- Testing
- Refactoring
- Pair programming
- Collective ownership
- Coding standards
- 40-hour week

The planning game

- Goal: Plan the next release, maximising value & controlling development costs and risk
- Pieces: user story cards

The planning game

- Goal: Plan the next release, maximising value & controlling development costs and risk
- Pieces: user story cards
- Players:
 - Customer understands scope, priority, business needs for releases: sorts cards by priority
 - Developers estimate risk and effort: sorts cards by risk, split cards if more than 2–4 weeks

The planning game

- Goal: Plan the next release, maximising value & controlling development costs and risk
- Pieces: user story cards
- Players:
 - Customer understands scope, priority, business needs for releases: sorts cards by priority
 - Developers estimate risk and effort: sorts cards by risk, split cards if more than 2–4 weeks
- Phases: Exploration, Commitment, Steer

On-site customer

- A customer — someone capable of making the business's decisions in the planning game — sits with the development team always ready to:

On-site customer

- A customer — someone capable of making the business's decisions in the planning game — sits with the development team always ready to:
 - Clarify

On-site customer

- A customer — someone capable of making the business's decisions in the planning game — sits with the development team always ready to:
 - Clarify
 - Write functional tests

On-site customer

- A customer — someone capable of making the business's decisions in the planning game — sits with the development team always ready to:
 - Clarify
 - Write functional tests
 - Make small-scale priority and scope decisions

On-site customer

- A customer — someone capable of making the business's decisions in the planning game — sits with the development team always ready to:
 - Clarify
 - Write functional tests
 - Make small-scale priority and scope decisions
- Customer maybe does their normal work when not needed to interact with the development team.

Small releases

- Release as frequently as is possible whilst still adding some business value in each release.

Small releases

- Release as frequently as is possible whilst still adding some business value in each release.
- This ensures:
 - You get feedback as soon as possible

Small releases

- Release as frequently as is possible whilst still adding some business value in each release.
- This ensures:
 - You get feedback as soon as possible
 - The customer has the most essential functionality as soon as possible.

Metaphor

- About an easily-communicated overarching view of system. E.g. "Desktop"

Metaphor

- About an easily-communicated overarching view of system. E.g. "Desktop"
- Encompasses concept of software architecture

Metaphor

- About an easily-communicated overarching view of system. E.g. "Desktop"
- Encompasses concept of software architecture
- Provides a sense of cohesion

Metaphor

- About an easily-communicated overarching view of system. E.g. "Desktop"
- Encompasses concept of software architecture
- Provides a sense of cohesion
- Often suggests a consistent vocabulary

Continuous integration

- Code is integrated, debugged and tested in full system build at most a few hours or one day after being written.

Continuous integration

- Code is integrated, debugged and tested in full system build at most a few hours or one day after being written.
- Maintains a working system at all times

Continuous integration

- Code is integrated, debugged and tested in full system build at most a few hours or one day after being written.
- Maintains a working system at all times
- Responsibility for integration failures easy to trace

Continuous integration

- Code is integrated, debugged and tested in full system build at most a few hours or one day after being written.
- Maintains a working system at all times
- Responsibility for integration failures easy to trace
- If integration difficult, maybe new feature was not understood well, so integration should be abandoned

Simple design

- Motto: do the simplest thing that could possibly work.
- Don't design for tomorrow: you might not need it.

Testing

- Any program feature without an automated test simply doesn't exist.

Testing

- Any program feature without an automated test simply doesn't exist.
- Test everything that could break.

Testing

- Any program feature without an automated test simply doesn't exist.
- Test everything that could break.
- Programmers write unit tests:
 - Use a good automated testing framework (e.g. JUnit) to minimise the effort of writing, running and checking tests.

Testing

- Any program feature without an automated test simply doesn't exist.
- Test everything that could break.
- Programmers write unit tests:
 - Use a good automated testing framework (e.g. JUnit) to minimise the effort of writing, running and checking tests.
- Customers (with developer help) write functional tests.

Refactoring

- Refactoring is especially vital for XP because of the way it dives almost straight into coding.

Refactoring

- Refactoring is especially vital for XP because of the way it dives almost straight into coding.
- Later redesign is essential.

Refactoring

- Refactoring is especially vital for XP because of the way it dives almost straight into coding.
- Later redesign is essential.
- "Three strikes and you refactor":
 1. The first time you need some piece of code you just write it.
 2. The second time, you are annoyed but probably duplicate it anyway.
 3. The third time, you refactor and use the shared code.

Refactoring

- Refactoring is especially vital for XP because of the way it dives almost straight into coding.
- Later redesign is essential.
- "Three strikes and you refactor":
 1. The first time you need some piece of code you just write it.
 2. The second time, you are annoyed but probably duplicate it anyway.
 3. The third time, you refactor and use the shared code.
- Do refactorings that you know are beneficial.

Pair programming

- All production code is written by two people at one machine.

Pair programming

- All production code is written by two people at one machine.
- You pair with different people on the team and take each role at different times.

Pair programming

- All production code is written by two people at one machine.
- You pair with different people on the team and take each role at different times.
- Two roles in each pair:
 - The one with the keyboard and the mouse is coding

Pair programming

- All production code is written by two people at one machine.
- You pair with different people on the team and take each role at different times.
- Two roles in each pair:
 - The one with the keyboard and the mouse is coding
 - The other partner thinks more strategically about:

Pair programming

- All production code is written by two people at one machine.
- You pair with different people on the team and take each role at different times.
- Two roles in each pair:
 - The one with the keyboard and the mouse is coding
 - The other partner thinks more strategically about:
 - Is this whole approach going to work?

Pair programming

- All production code is written by two people at one machine.
- You pair with different people on the team and take each role at different times.
- Two roles in each pair:
 - The one with the keyboard and the mouse is coding
 - The other partner thinks more strategically about:
 - Is this whole approach going to work?
 - What are some other test cases that might not work yet?

Pair programming

- All production code is written by two people at one machine.
- You pair with different people on the team and take each role at different times.
- Two roles in each pair:
 - The one with the keyboard and the mouse is coding
 - The other partner thinks more strategically about:
 - Is this whole approach going to work?
 - What are some other test cases that might not work yet?
 - Is there some way to simplify the whole system so the current problem just disappears?

Collective ownership

- You don't have "your modules" which no-one else is allowed to touch.

Collective ownership

- You don't have "your modules" which no-one else is allowed to touch.
- If anybody sees a way to improve the design of the whole system they don't need anyone else's permission to go ahead and make all the necessary changes.

Collective ownership

- You don't have "your modules" which no-one else is allowed to touch.
- If anybody sees a way to improve the design of the whole system they don't need anyone else's permission to go ahead and make all the necessary changes.
- A good configuration management tool is vital.

Coding standards

- The whole team adheres to a single set of conventions about how code is written.
- This is needed to make pair programming and collective ownership work.

Sustainable pace

- Also known as "40-hour week" — but this means not 60, rather than not 35!
- People need to be fresh, creative, careful and confident to work effectively in the way XP prescribes.
- There might be a week coming up to deadlines when people had to work more than this, but there shouldn't be two consecutive such weeks.

Mix and match?

- Can you use just some of the XP practices?

Mix and match?

- Can you use just some of the XP practices?
- Maybe... but they are very interrelated, so it's dangerous.

Mix and match?

- Can you use just some of the XP practices?
- Maybe... but they are very interrelated, so it's dangerous.
- If you do collective ownership but not coding standards — the code will end up a mess

Mix and match?

- Can you use just some of the XP practices?
- Maybe... but they are very interrelated, so it's dangerous.
- If you do collective ownership but not coding standards — the code will end up a mess
- If you do simple design but not refactoring — you'll get stuck!

Mix and match?

- Can you use just some of the XP practices?
- Maybe... but they are very interrelated, so it's dangerous.
- If you do collective ownership but not coding standards — the code will end up a mess
- If you do simple design but not refactoring — you'll get stuck!
- XP practices support each other in many ways.

How XP practices support each other

- All twelve XP practices are heavily interrelated and mutually reinforcing.



From Extreme Programming Explained: Embrace Change by Kent Beck.

Where is XP applicable?

- The scope of situations in which XP is appropriate is somewhat controversial.

Where is XP applicable?

- The scope of situations in which XP is appropriate is somewhat controversial.
- Documented cases where it has worked well for in-house custom software development (e.g. Chrysler).

Where is XP applicable?

- The scope of situations in which XP is appropriate is somewhat controversial.
- Documented cases where it has worked well for in-house custom software development (e.g. Chrysler).
- A decade ago it seemed clear it wouldn't work for Microsoft: big releases were essential to the business. Now we have automated OS updates, and Microsoft is a Gold Sponsor of an Agile conference.

Where is XP applicable?

- The scope of situations in which XP is appropriate is somewhat controversial.
- Documented cases where it has worked well for in-house custom software development (e.g. Chrysler).
- A decade ago it seemed clear it wouldn't work for Microsoft: big releases were essential to the business. Now we have automated OS updates, and Microsoft is a Gold Sponsor of an Agile conference.
- XP does need: team in one place, customer on site, etc.

Scrum

- "A framework within which people can address complex adaptive problems, while productively and creatively delivering products of the highest possible value"

Scrum

- "A framework within which people can address complex adaptive problems, while productively and creatively delivering products of the highest possible value"
- People: First mentioned by Hirotaka Takeuchi and Ikujiro Nonaka; developed by Ken Schwaber and Jeff Sutherland (Agile Manifesto signees) since the 90s, first published 1995.
- More info: The Scrum Guide ([scrumguides.org](https://www.scrumguides.org)), published and revised 6 times by Schwaber and Sutherland.

Scrum values

- Commitment to achieve team goals
- Courage: doing the right thing, tackling problems
- Focus on work and goals of the team
- Openness about work and its challenges
- Respect towards members of the team as capable, independent people

The Scrum team

- Self-organising: chooses how to do the work, is not told how

The Scrum team

- Self-organising: chooses how to do the work, is not told how
- Cross-functional: have the needed competencies without need of help from outside the team
- Team members:
 - Product Owner
 - Development Team
 - Scrum Master

The Product Owner (PO)

- One person responsible for maximising the value of the team's work through the management of the Product Backlog (ordered list of requirements for the software).

The Product Owner (PO)

- One person responsible for maximising the value of the team's work through the management of the Product Backlog (ordered list of requirements for the software).
- Responsibilities:
 - Clearly expressing items in the Product Backlog

The Product Owner (PO)

- One person responsible for maximising the value of the team's work through the management of the Product Backlog (ordered list of requirements for the software).
- Responsibilities:
 - Clearly expressing items in the Product Backlog
 - Maintaining visibility, transparency, clarity of Product Backlog

The Product Owner (PO)

- One person responsible for maximising the value of the team's work through the management of the Product Backlog (ordered list of requirements for the software).
- Responsibilities:
 - Clearly expressing items in the Product Backlog
 - Maintaining visibility, transparency, clarity of Product Backlog
 - Ensuring the team's understanding of the Product Backlog

The Product Owner (PO)

- One person responsible for maximising the value of the team's work through the management of the Product Backlog (ordered list of requirements for the software).
- Responsibilities:
 - Clearly expressing items in the Product Backlog
 - Maintaining visibility, transparency, clarity of Product Backlog
 - Ensuring the team's understanding of the Product Backlog
 - Ordering items

The Product Owner (PO)

- One person responsible for maximising the value of the team's work through the management of the Product Backlog (ordered list of requirements for the software).
- Responsibilities:
 - Clearly expressing items in the Product Backlog
 - Maintaining visibility, transparency, clarity of Product Backlog
 - Ensuring the team's understanding of the Product Backlog
 - Ordering items
 - Optimizing the value of the work of the team

The Product Owner (PO)

- One person responsible for maximising the value of the team's work through the management of the Product Backlog (ordered list of requirements for the software).
- Responsibilities:
 - Clearly expressing items in the Product Backlog
 - Maintaining visibility, transparency, clarity of Product Backlog
 - Ensuring the team's understanding of the Product Backlog
 - Ordering items
 - Optimizing the value of the work of the team
- This work can be delegated to the Development Team, but PO remains accountable.

The Development Team

- Should be small: 3–9 members
- Responsible for producing a potentially releasable increment of the software which is considered "Done".
- No titles, no sub-teams, all equally accountable.

The Scrum Master

- Servant-leader to the Scrum team.

The Scrum Master

- Servant-leader to the Scrum team.
- Promotes and supports Scrum as defined in the Scrum Guide, e.g. by coaching, reviewing the use of Scrum practices, facilitating Scrum events.

The Scrum Master

- Servant-leader to the Scrum team.
- Promotes and supports Scrum as defined in the Scrum Guide, e.g. by coaching, reviewing the use of Scrum practices, facilitating Scrum events.
- Ensures events are held, purpose is understood, they keep to time.

The Scrum Master

- Servant-leader to the Scrum team.
- Promotes and supports Scrum as defined in the Scrum Guide, e.g. by coaching, reviewing the use of Scrum practices, facilitating Scrum events.
- Ensures events are held, purpose is understood, they keep to time.
- Facilitates helpful interactions between those outside the Scrum team and the Scrum team.

The Sprint

- Time-boxed constant duration event of maximum one month during which a new increment is developed up to the point of "Done".

The Sprint

- Time-boxed constant duration event of maximum one month during which a new increment is developed up to the point of "Done".
- Scrum consists of a succession of sprints.

The Sprint

- Time-boxed constant duration event of maximum one month during which a new increment is developed up to the point of "Done".
- Scrum consists of a succession of sprints.
- Each sprint has a Sprint Goal and a Sprint Backlog (subset of Product Backlog).

The Sprint

- Time-boxed constant duration event of maximum one month during which a new increment is developed up to the point of "Done".
- Scrum consists of a succession of sprints.
- Each sprint has a Sprint Goal and a Sprint Backlog (subset of Product Backlog).
- Container for all other Scrum events:
 - Sprint Planning
 - Daily Scrums
 - Sprint Review
 - Sprint Retrospective

Sprint planning

- Time-boxed (max 8 hours per one month sprint) event deciding on the work to be performed:

Sprint planning

- Time-boxed (max 8 hours per one month sprint) event deciding on the work to be performed:
- What the sprint can deliver:
 - The Sprint Goal — based on objectives put forward by the PO and discussion by the whole team
 - The items from the Product Backlog that would achieve the Sprint Goal — brought up by the PO but the decision of the Development Team
 - A plan of the work (usually split into units of max 1 day each) — discussed by the Development Team, with PO potentially proposing trade-offs

Sprint planning

- Time-boxed (max 8 hours per one month sprint) event deciding on the work to be performed:
- What the sprint can deliver:
 - The Sprint Goal — based on objectives put forward by the PO and discussion by the whole team
 - The items from the Product Backlog that would achieve the Sprint Goal — brought up by the PO but the decision of the Development Team
 - A plan of the work (usually split into units of max 1 day each) — discussed by the Development Team, with PO potentially proposing trade-offs
- Items + plans moved to Sprint Backlog document.

Daily Scrums

- Daily time-boxed 15-minute event held in same place at the same time.

Daily Scrums

- Daily time-boxed 15-minute event held in same place at the same time.
- Development Team plans next 24 hours.

Daily Scrums

- Daily time-boxed 15-minute event held in same place at the same time.
- Development Team plans next 24 hours.
- Aims: inspecting progress with Sprint Backlog towards Sprint Goal.

Daily Scrums

- Daily time-boxed 15-minute event held in same place at the same time.
- Development Team plans next 24 hours.
- Aims: inspecting progress with Sprint Backlog towards Sprint Goal.
- One possible structure — each person stating:
 - Progress towards the goal in the last 24 hours
 - Plans for the next 24 hours
 - Potential impediments

Daily Scrums — advantages

- Improves communication
- Reduces need for other meetings
- Identifies impediments and who can help (separate longer meetings)
- Promotes quick decision making
- Improves level of knowledge of the Development Team

Sprint review

- Time-boxed (max 4-hour for one month sprint) informal event held at end of sprint.

Sprint review

- Time-boxed (max 4-hour for one month sprint) informal event held at end of sprint.
- Attended by Scrum team and stakeholders invited by PO.

Sprint review

- Time-boxed (max 4-hour for one month sprint) informal event held at end of sprint.
- Attended by Scrum team and stakeholders invited by PO.
- Aims:
 - Inspecting increment (what was "Done" and what not, what went well, problems, solutions)
 - Potentially adapting Product Backlog in view of progress
 - Collaborating on ways forward to optimize value
 - Gathering feedback from stakeholders

Sprint review

- Time-boxed (max 4-hour for one month sprint) informal event held at end of sprint.
- Attended by Scrum team and stakeholders invited by PO.
- Aims:
 - Inspecting increment (what was "Done" and what not, what went well, problems, solutions)
 - Potentially adapting Product Backlog in view of progress
 - Collaborating on ways forward to optimize value
 - Gathering feedback from stakeholders
- Outcome: revised Product Backlog, probable items for the next sprint.

Sprint retrospective

- Time-boxed (max 3-hour meeting for one month sprint) event held after Sprint Review and before next Sprint Planning.

Sprint retrospective

- Time-boxed (max 3-hour meeting for one month sprint) event held after Sprint Review and before next Sprint Planning.
- Aims:
 - Review last sprint with regards to people, relationships, process and tools
 - Identify and order what went well and potential improvements
 - Create a plan for carrying out improvements and potentially adapting definition of "Done" for next sprint

Sprint retrospective

- Time-boxed (max 3-hour meeting for one month sprint) event held after Sprint Review and before next Sprint Planning.
- Aims:
 - Review last sprint with regards to people, relationships, process and tools
 - Identify and order what went well and potential improvements
 - Create a plan for carrying out improvements and potentially adapting definition of "Done" for next sprint
- Scrum Master acts as a peer team member, keeps meeting positive and productive.

Applicability of Scrum

- Scrum is currently the most popular and widely used agile framework.
- Companies using Scrum: Google, Apple, Facebook, Amazon, IBM, Intel, Netflix, Adobe, ING, Vodafone.
- Reported benefits (Scrum Alliance): Better reactivity to change, better alignment business–IT, faster time to market.
- Scrum also used in research, sales, marketing and advanced technologies.

Kanban

- "A strategy for optimizing the flow of value through a process that uses a visual, pull-based system" (Kanban Guide)
- People: Taiichi Ohno and W. Edwards Deming from Toyota and the Toyota Production System.
- David J. Anderson from Microsoft made it applicable to any company that needs organisation, since 2004.
- More info: The Kanban Guide (kanbanguides.org)

Kanban theory

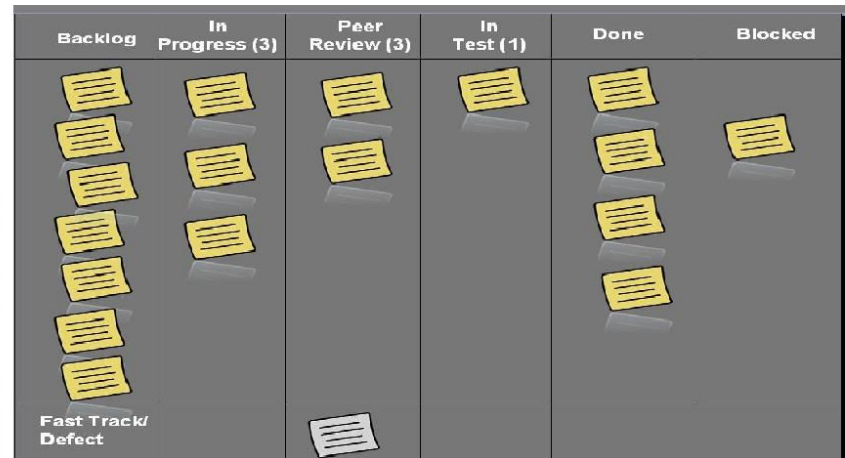
- Kanban uses flow theory.
- Flow = "movement of potential value through a system" (The Kanban Guide)
- Strategy: optimise value by optimising flow, finding the right balance between:
 - Effectiveness: delivering what customers want when they want it
 - Efficiency: allocating resources optimally to produce value
 - Predictability: ability to forecast value delivery accurately in the presence of uncertainty

Kanban practices

- Visualising the workflow
- Limiting work in progress (WIP)
- Actively managing items in the workflow
- Improving the workflow

Visualising the workflow

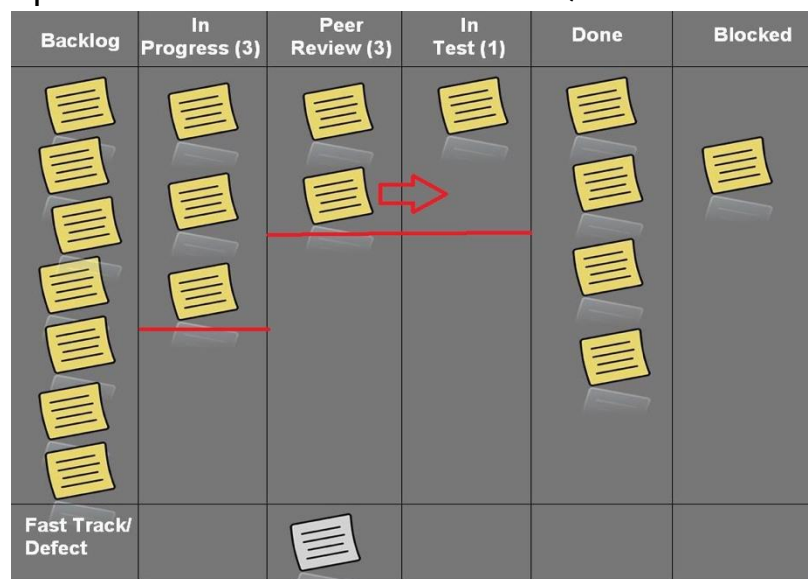
- Use a Kanban board (physical or digital) to visualise and keep workflow transparent amongst team members.
- Make the invisible visible.



- Useful tools for Kanban boards: Jira and Trello.

Limit work in progress (WIP)

- WIP = any work items between a started point and a finished point, i.e. started but not finished. Also used to indicate their number.
- Little's Law: The more things (WIP) you work on at the same time (on average), the longer it will take (on average).
- Can lower WIP in single column, several grouped columns, or whole board.
- Side effect: creates pull system, improves collective focus, commitment, and collaboration.



Actively managing items in the workflow

- Limiting WIP helps achieve flow, but is not sufficient. Additional management could include:
- Avoiding work items piling up in part of the workflow
- Ensuring work items pulled at about the same rate as leaving the workflow
- Ensuring work items do not age unnecessarily (using the service level expectation / SLE)
- Responding quickly to unblock blocked work

Improving the workflow

- Continuously improve workflow towards better balance of effectiveness–efficiency–predictability.
- Alterations should be just-in-time as context dictates, not waiting for formal meetings.
- Useful: the use of measures and visualisations.

Kanban measures

- Four mandatory flow measures:
 - WIP
 - Throughput: The number of work items finished per unit of time
 - Work Item Age: Time elapsed between start of work item and current time
 - Cycle Time: Time elapsed between start and finish of a work item
- Can help reach shared understanding of Kanban system's health and performance.
- Useful to use charts to represent them, and use them to inform the Kanban practices.

Applicability of Kanban

- Companies using Kanban or Kanban tools: HP, Spotify, Pixar Studios, Zara, Pirelli.
- Kanban is often used to enhance Scrum. See [The Kanban Guide for Scrum Teams](#).
- Also used in finance, marketing, healthcare.

Reading

- On XP:
 - Essential: Browse through the tutorial at extremeprogramming.org
 - Recommended: Sommerville ESP Chapter 2 Section 2.2
- On Scrum:
 - Essential: The Scrum Guide — download from scrumguides.org
 - Recommended: Sommerville ESP Chapter 2 Section 2.3
- On Kanban:
 - Essential: YouTube introduction to Kanban (youtube.com/watch?v=iVaFVa7HYj4)
 - Essential: The Kanban Guide — download from kanbanguides.org

Reading (continued)

- Essential: "Revisiting the Principles and General Practices of the Kanban Method" by David J. Anderson (dja.com)
- Recommended: Kanban Guide for Scrum Teams — download from scrum.org
- Recommended: YouTube JIRA tutorial for Kanban boards
- Recommended: YouTube Trello tutorial for Kanban boards