Plan-driven and Agile Methodologies

Instructor
Battista Biggio

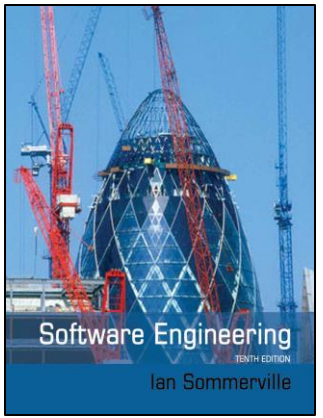
M.Sc. in Computer Engineering, Cybersecurity and Artificial Intelligence
University of Cagliari, Italy

Topics Covered

- Software Process Models: Plan-driven and Agile Methodologies
- Agile Manifesto and Principles
- Agile Development Techniques (XP)
- Agile Project Management (Scrum, Kanban)
- Problems when Scaling Agile Methods
- What's Next? DevOps and Continuous Delivery

Main references / materials accompanying this lecture

- *Software Engineering, Ian Sommerville (eds. 9, 10), Ch. 2 -3 - (book available online)*
- *Further readings / tutorials at the end of the slides*

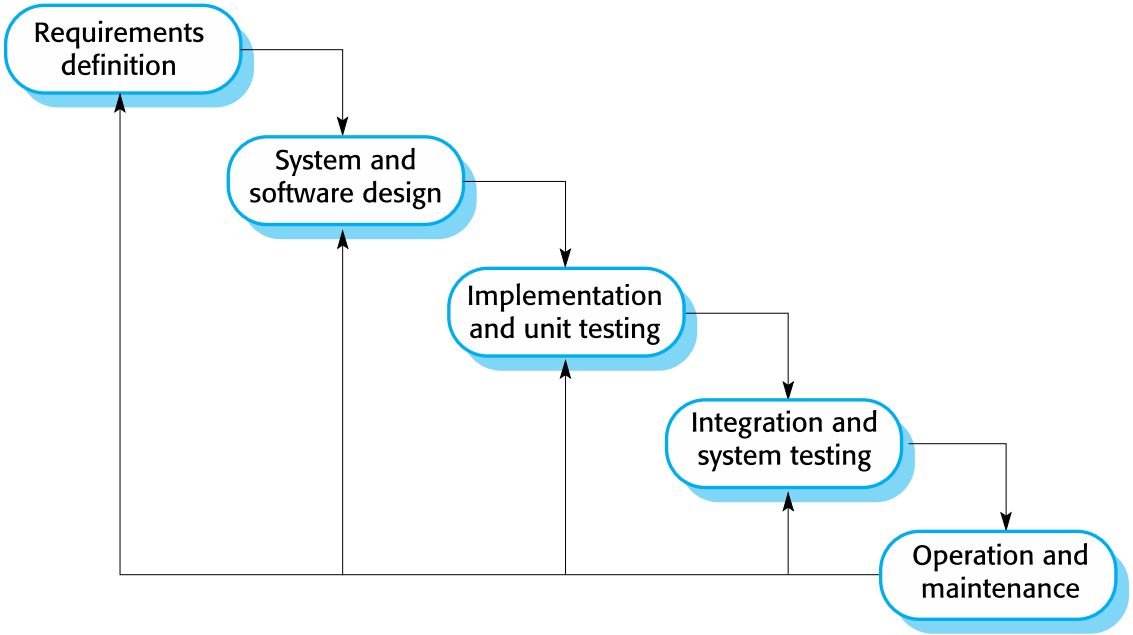


Software Process Models

Software Process Models

- The waterfall model
  - Plan-driven model. Separate and distinct phases of specification and development.
- Incremental development
  - Specification, development and validation are interleaved. May be plan-driven or agile.
- Integration and configuration
  - The system is assembled from existing configurable components. May be plan-driven or agile.
- In practice, most large systems are developed using a process that incorporates elements from all of these models

The Waterfall Model



Waterfall Model Problems

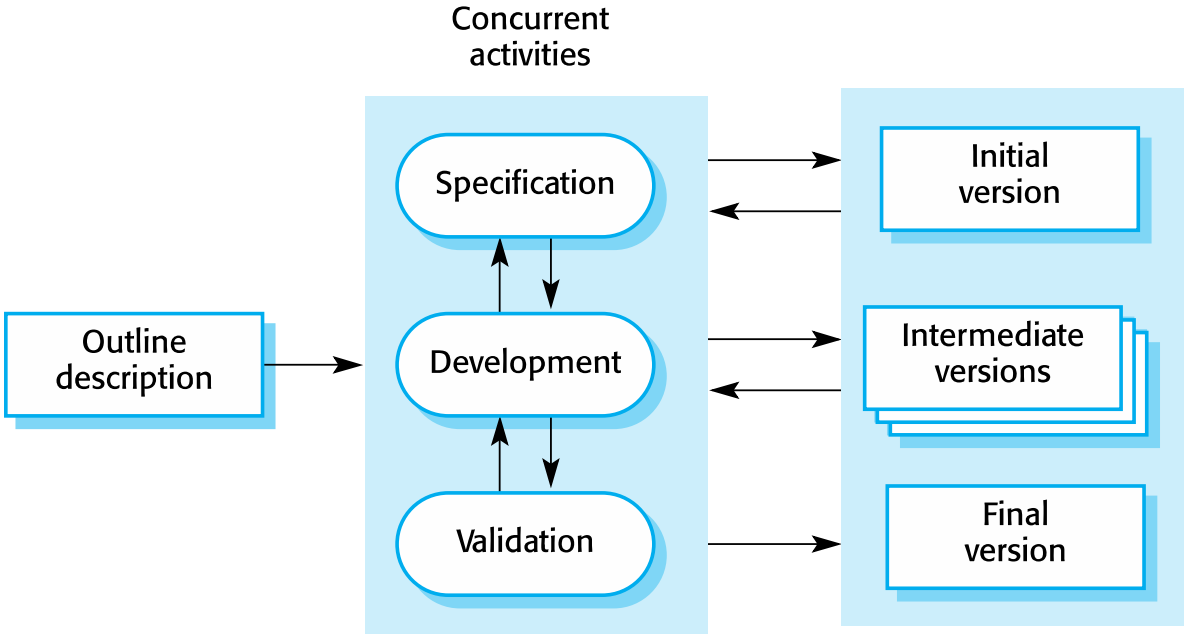
- **Main drawback:** difficulty of accommodating change after the process is underway
  - document-driven approach requiring each phase to be completed before moving onto the next

- Inflexible partitioning of the project into distinct stages makes it difficult to respond to changing customer requirements
  - appropriate if requirements are well-understood and changes are limited during design
  - few business systems have stable requirements

- The waterfall model is mostly used for large systems engineering projects where a system is developed at several sites
  - its plan-driven nature helps coordinate the work

Incremental Development

- **Main goal**: to incorporate customer feedback more easily *during* the development process



Incremental Development Benefits

- The cost of accommodating changing customer requirements is reduced
  - The amount of analysis and documentation that has to be redone is much less than that required by the waterfall model

- **Easier to incorporate customer feedback:** customers can comment on demonstrations of the software and propose changes/modifications

- More rapid delivery and deployment of useful software to the customer is possible
  - Customers are able to use and gain value from the software earlier than is possible with a waterfall process

Incremental Development Problems

- The process is not visible
  - Managers need regular deliverables to measure progress. If systems are developed quickly, it is not cost-effective to produce documents that reflect every version of the system

- System structure tends to degrade as new increments are added
  - Unless time and money is spent on refactoring to improve the software, regular change tends to corrupt its structure
  - Incorporating further software changes becomes increasingly difficult and costly

Integration and Configuration (Software Reuse)

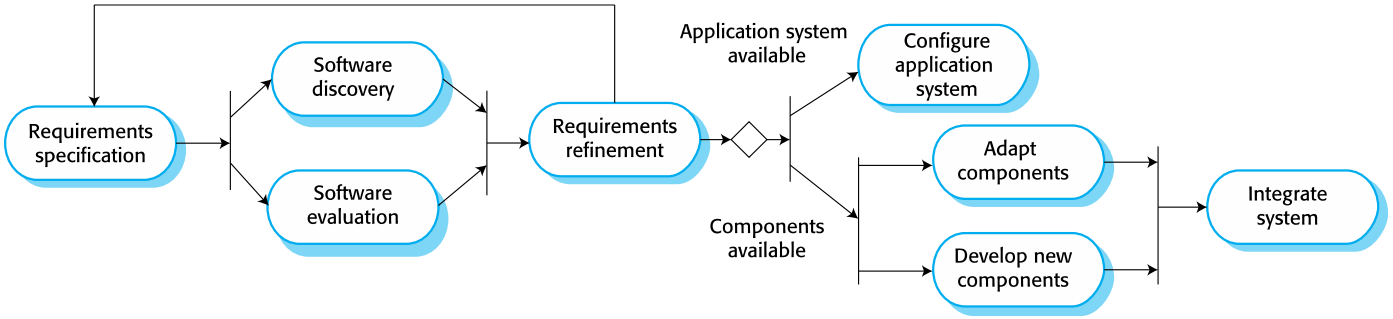
- **Main idea:** *software reuse*
  - systems are integrated from existing components or application systems

- Reused elements may be configured to adapt their behaviour and functionality to a user's requirements

- Main examples:
  - Open-source third-party libraries (Numpy, Scipy, OpenCV, Tensorflow, PyTorch, etc.)
  - Web services that are developed according to service standards, available for remote invocation (e.g., Google Cloud Vision APIs)

- Reuse is one of the standard approaches for building many types of business system
  - Think of commercial software products built on open-source software...

Reuse-oriented Software Engineering



- **Advantages and Disadvantages**
  - Reduced costs and risks as less software is developed from scratch
  - Faster delivery and deployment of system
  - Potential compromise of requirements / system may not meet real needs of users
  - Loss of control over evolution of reused system elements

Lessons Learned

- Examples of main software processes
  - The waterfall model
    - Plan-driven model. Separate and distinct phases of specification and development.
  - Incremental development
    - Specification, development and validation are interleaved. May be plan-driven or agile.
  - Integration and configuration (*software reuse*)
    - The system is assembled from existing configurable components. May be plan-driven or agile.

- In practice, most large systems are developed using a process that incorporates elements from all of these models

Agile Manifesto and Principles

Rapid Software Development

- Rapid development and delivery may be a very important requirement for software systems
  - Businesses operate in a fast–changing environment
  - It is practically impossible to produce a set of stable software requirements
  - Software has to evolve quickly to reflect changing business needs

- Plan-driven development (e.g., *waterfall*) is essential for some types of system
  - but it may not meet more flexible business needs

- *Agile development methods* emerged in the late 1990s
  - **Goal:** to radically reduce the delivery time for working software systems

Agile Development

- Program specification, design and implementation are interleaved
- The system is developed as a series of versions or increments with stakeholders involved in version specification and evaluation
- Frequent delivery of new versions for evaluation
  - release every 2-4 weeks (with a reduced set of features which grows/changes incrementally)
- Automated testing tools to support development
- Minimal documentation – focus on working code

Plan-driven vs Agile Development

- Plan-driven development
  - separate development stages, each with outputs planned in advance
  - not necessarily waterfall model – plan-driven, incremental development is possible
  - iteration occurs within activities

- Agile development
  - specification, design, implementation and testing are interleaved
  - the outputs from the development process are decided through a process of negotiation during the software development process

Agile Methods

- Dissatisfaction with the overheads involved in software design methods of the 1980s and 1990s led to the creation of agile methods

- **These methods:**
  - Focus on the code rather than the design
  - Are based on an iterative approach to software development
  - Are intended to deliver working software quickly and evolve this quickly to meet changing requirements

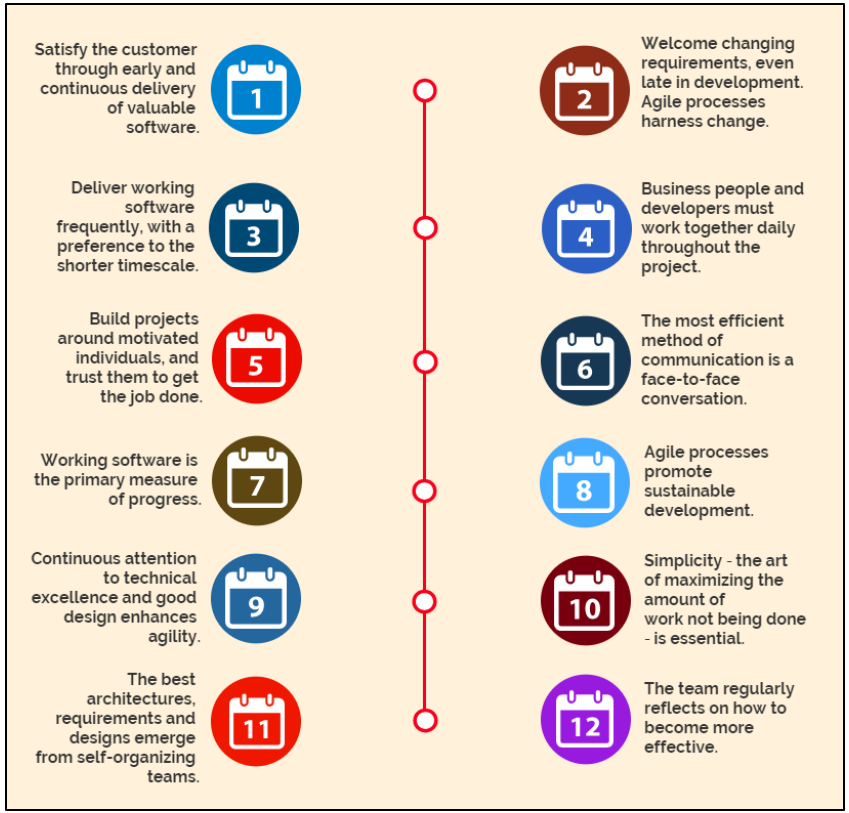
- **The aim of agile methods is**
  - to reduce overheads in the software process (e.g. by limiting documentation)
  - to be able to respond quickly to changing requirements without excessive rework

Agile Manifesto

- *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.*

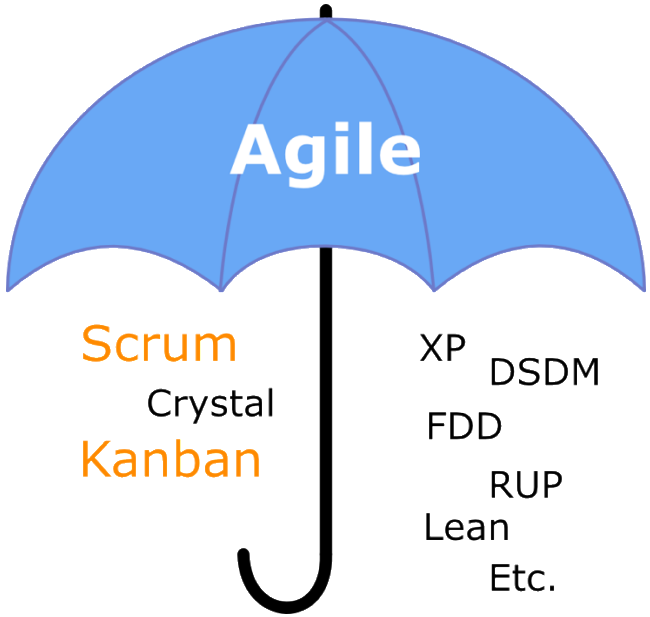
The 12 Agile Principles



A Compact View of the Agile Principles

Principle	Description
Customer involvement	Customers should be closely involved throughout the development process. Their role is to provide and prioritize new system requirements and to evaluate the iterations of the system.
Incremental delivery	The software is developed in increments with the customer specifying the requirements to be included in each increment.
People not process	The skills of the development team should be recognized and exploited. Team members should be left to develop their own ways of working without prescriptive processes.
Embrace change	Expect the system requirements to change and so design the system to accommodate these changes.
Maintain simplicity	Focus on simplicity in both the software being developed and in the development process. Wherever possible, actively work to eliminate complexity from the system.

Agile Umbrella



When to Use Agile Methods

- Product development where a software company is developing a small or medium-sized product for sale
  - Many software products and apps are developed using an *agile* approach

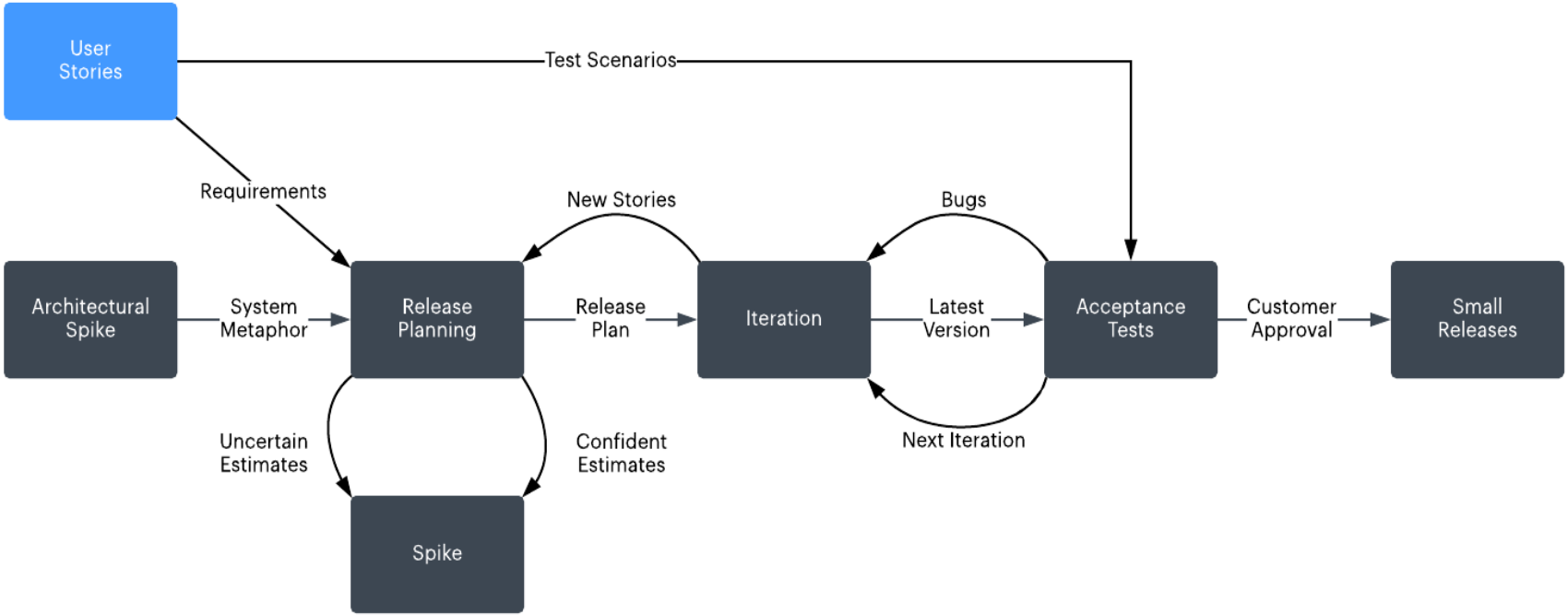
- Custom system development within an organization
  - clear commitment from the customer to become involved in the development process
  - few external rules and regulations that affect the software

Agile Development Techniques

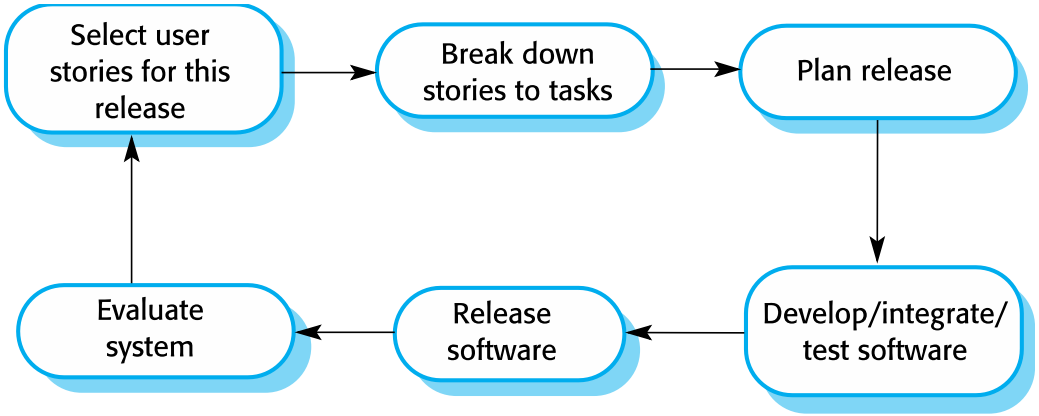
Extreme Programming (XP)

- A very influential agile method, developed in the late 1990s, that introduced a range of agile development techniques
- **Goal:** to improve software quality and responsiveness to changing customer requirements
- Extreme Programming (XP) takes an *extreme* approach to iterative development
  - New versions may be built several times per day
  - Increments are delivered to customers every 2 weeks
  - All tests must be run for every build
  - The build is only accepted if tests run successfully

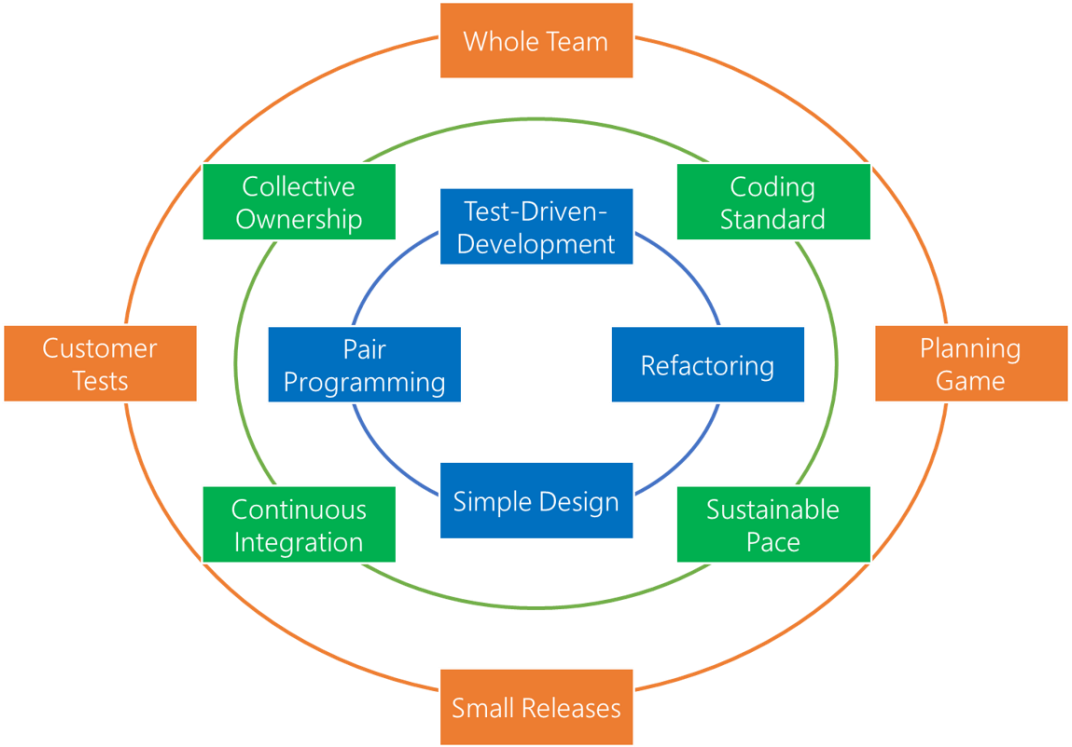
XP Release Cycle in a Nutshell



XP Release Cycle (simplified)



XP Practices



XP Practices

Principle or practice	Description
Incremental Planning (Planning Game)	Requirements are recorded on story cards and the stories to be included in a release are determined by the time available and their relative priority. The developers break these stories into development 'Tasks'
Small releases	The minimal useful set of functionality that provides business value is developed first. Releases of the system are frequent and incrementally add functionality to the first release
On-site customer (Requirements/Testing)	A representative of the end-user of the system (the customer) should be available full time for the use of the XP team. In an extreme programming process, the customer is a member of the development team and is responsible for bringing system requirements to the team for implementation

XP Practices

Principle or practice	Description
Simple design	Enough design is carried out to meet the current requirements and no more
Test-first development	An automated unit test framework is used to write tests for a new piece of functionality before that functionality itself is implemented
Refactoring	All developers are expected to refactor the code continuously as soon as possible code improvements are found. This keeps the code simple and maintainable
Pair programming	Developers work in pairs, checking each other's work and providing the support to always do a good job

XP Practices

Principle or practice	Description
Collective ownership	The pairs of developers work on all areas of the system, so that no islands of expertise develop and all the developers take responsibility for all of the code. Anyone can change anything.
Continuous integration	As soon as the work on a task is complete, it is integrated into the whole system. After any such integration, all the unit tests in the system must pass.
Sustainable pace	Large amounts of overtime are not considered acceptable as the net effect is often to reduce code quality and medium term productivity

XP and Agile Principles

- Incremental development is supported through small, frequent system releases
- Customer involvement means full-time customer engagement with the team
- People not process through pair programming, collective ownership and a process that avoids long working hours
- Change supported through regular system releases
- Maintaining simplicity through constant refactoring of code

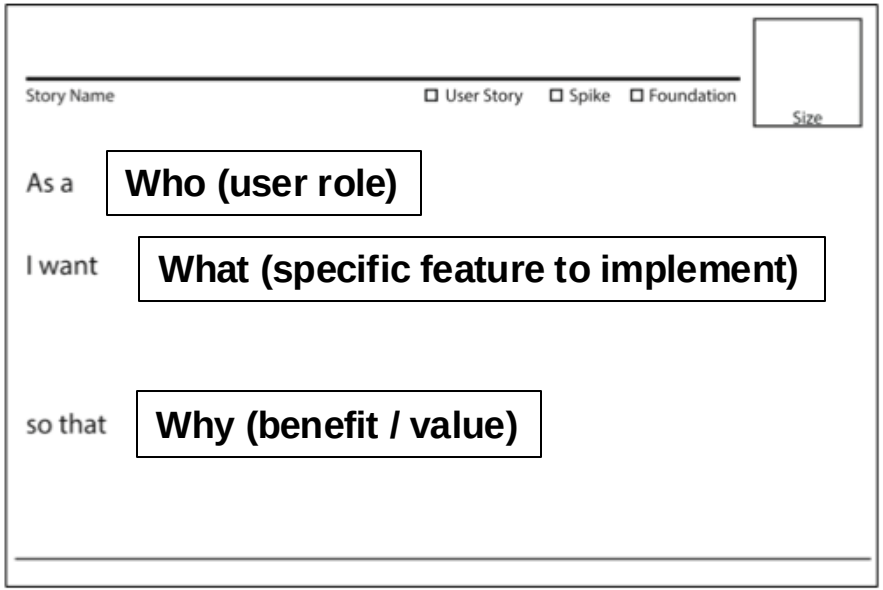
Influential XP Practices

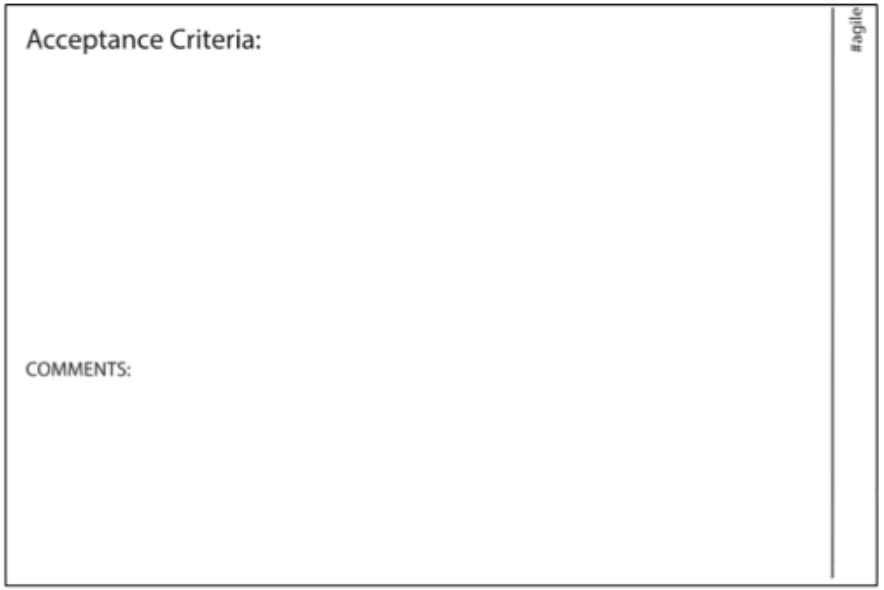
- XP has a technical focus and is not easy to integrate with management practice in most organizations
- Consequently, while agile development uses practices from XP, the method as originally defined is not widely used
- Key practices
  - User stories for specification
  - Refactoring
  - Test-first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

User Stories for Requirements

- In XP, a customer or user is part of the XP team and is responsible for making decisions on requirements
- User requirements are expressed as user stories or scenarios
- These are written on cards and the development team break them down into implementation tasks. These tasks are the basis of schedule and cost estimates
- The customer chooses the stories for inclusion in the next release based on their priorities and the schedule estimates

User Story Cards





User Story Cards

Backlog: Android app
Theme: Menu
MEN1

As **mobile user**
I want to **use an intuitive menu**
So I can **navigate around the application**

Backlog: Android app
Theme: Menu
MEN2

As **administrator**
I want to **be able to change the navigation structure**
So I can **keep the app fresh and prioritise key features remotely**

Backlog: Android app
Theme: Startup
STA1

As **mobile user**
I want to **see a startup screen**
So I can **realise the app is loading**

Refactoring

- Conventional wisdom in software engineering is to design for change
  - It is worth spending time and effort anticipating changes as this reduces costs later in the life cycle
- XP, however, suggests that this is not worthwhile as changes cannot be reliably anticipated
- Rather, it proposes constant code improvement (refactoring) to make changes easier when they have to be implemented

Refactoring

- Software improvements are made even if there is no immediate need for them
- Software is more understandable / easier to document and use
- Changes are easier to make because the code is well-structured and clear
- **Main drawback:** some changes requires architecture refactoring (much more expensive!)

Examples of Refactoring

- Re-organization of a class hierarchy to remove duplicate code
- Renaming attributes and methods to make them easier to understand
- Replacement of inline code with calls to methods that have been included in a program library

Test-first Development

- **Testing is central to XP:** the program is tested after every change has been made

- **XP testing features:**
  - Test-first development
  - Incremental test development from scenarios
  - User involvement in test development and validation
  - Automated tests are run each time that a new release is built (continuous integration)

Test-Driven Development (TDD)

- Writing tests before code clarifies the requirements to be implemented
- Tests are written as programs rather than data so that they can be executed automatically
- The test includes a check that it has executed correctly
  - Usually relies on a testing framework such as JUnit / Python unittest
- All tests are run automatically when new functionality is added to check that no errors are introduced

Customer Involvement

- The role of the customer in the testing process is to help develop acceptance tests for the stories to be implemented in the next release of the system
- The customer who is part of the team writes tests as development proceeds. All new code is therefore validated to ensure that it is what the customer needs
- However, people adopting the customer role have limited time available and so cannot work full-time with the development team
  - They may feel that providing the requirements was enough of a contribution and so may be reluctant to get involved in the testing process

Test Automation

- Tests are written as executable components before the task is implemented. Tests should:
  - be stand-alone
  - simulate the submission of input to be tested
  - check that the result meets the output specification

- An automated test framework (e.g. JUnit / Unittest) is a system that makes it easy to write executable tests and submit a set of tests for execution

- As testing is automated, there is always a set of tests that can be quickly and easily executed
  - Whenever any functionality is added to the system, the tests can be run and problems that the new code has introduced can be caught immediately

Problems with Test-first Development

- Programmers prefer programming to testing and sometimes they take short cuts when writing tests
  - **Potential issue:** incomplete tests that do not check for all possible exceptions that may occur

- Some tests can be very difficult to write incrementally
  - **Potential issue:** in a complex user interface, it is often difficult to write unit tests for the code that implements the 'display logic' and workflow between screens

- It difficult to judge the completeness of a set of tests. Although you may have a lot of system tests, your test set may not provide complete coverage.

Pair Programming

- Pair programming involves programmers working in pairs, developing code together
- This helps develop common ownership of code and spreads knowledge across the team
- It serves as an informal review process as each line of code is looked at by more than 1 person
- It encourages refactoring as the whole team can benefit from improving the system code

Pair Programming

- Programmers sit together at the same computer to develop the software
- Pairs are created dynamically so that all team members work with each other during the development process
- The sharing of knowledge that happens during pair programming is very important as it reduces the overall risks to a project when team members leave
- Pair programming is not necessarily inefficient and there is some evidence that suggests that a pair working together is more efficient than 2 programmers working separately

Agile Project Management

Agile Project Management

- **Goal:** to deliver software on time and within the planned budget
- **Standard approach:** plan-driven
  - Managers draw up a plan for the project showing what should be delivered, when it should be delivered and who will work on the development of the project deliverables
- **Agile project management** is different
  - It adapts to incremental development and to the practices used in agile methods
- Two main approaches described in this lecture: **Scrum** and **Kanban**

Agile Project Management with Scrum

Scrum

- Scrum is an agile method that focuses on managing iterative development rather than specific agile practices.

- There are three main phases in Scrum
  1. **Outline Planning:** sets the general objectives for the project and design the software architecture
  2. **Sprint Cycles:** each cycle develops an increment of the system
  3. **Project Closure:** wraps up the project, completes required documentation such as system help frames and user manuals and assesses the lessons learned from the project.

Scrum Terminology

Scrum term	Definition
Development team	A self-organizing group of software developers, which should be no more than 7 people. They are responsible for developing the software and other essential project documents.
Potentially shippable product increment	The software increment that is delivered from a sprint. The idea is that this should be 'potentially shippable' which means that it is in a finished state and no further work, such as testing, is needed to incorporate it into the final product. In practice, this is not always achievable.
Product backlog	This is a list of 'to do' items which the Scrum team must tackle. They may be feature definitions for the software, software requirements, user stories or descriptions of supplementary tasks that are needed, such as architecture definition or user documentation.
Product owner	An individual (or possibly a small group) whose job is to identify product features or requirements, prioritize these for development and continuously review the product backlog to ensure that the project continues to meet critical business needs. The Product Owner can be a customer but might also be a product manager in a software company or other stakeholder representative.

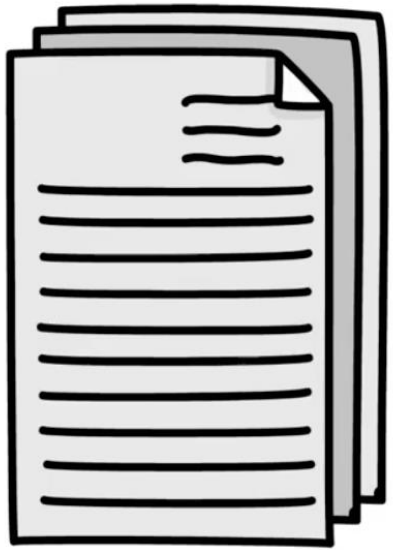
Scrum Terminology

Scrum term	Definition
Scrum	A daily meeting of the Scrum team that reviews progress and prioritizes work to be done that day. Ideally, this should be a short face-to-face meeting that includes the whole team.
ScrumMaster	The ScrumMaster is responsible for ensuring that the Scrum process is followed and guides the team in the effective use of Scrum. He or she is responsible for interfacing with the rest of the company and for ensuring that the Scrum team is not diverted by outside interference. The Scrum developers are adamant that the ScrumMaster should not be thought of as a project manager. Others, however, may not always find it easy to see the difference.
Sprint	A development iteration. Sprints are usually 2-4 weeks long.
Velocity	An estimate of how much product backlog effort that a team can cover in a single sprint. Understanding a team's velocity helps them estimate what can be covered in a sprint and provides a basis for measuring improving performance.

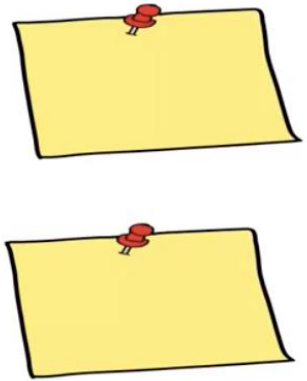
The 3 Key Roles

The 3 Key Artifacts (Docs)

Product Backlog



Sprint Backlog



Burndown Chart

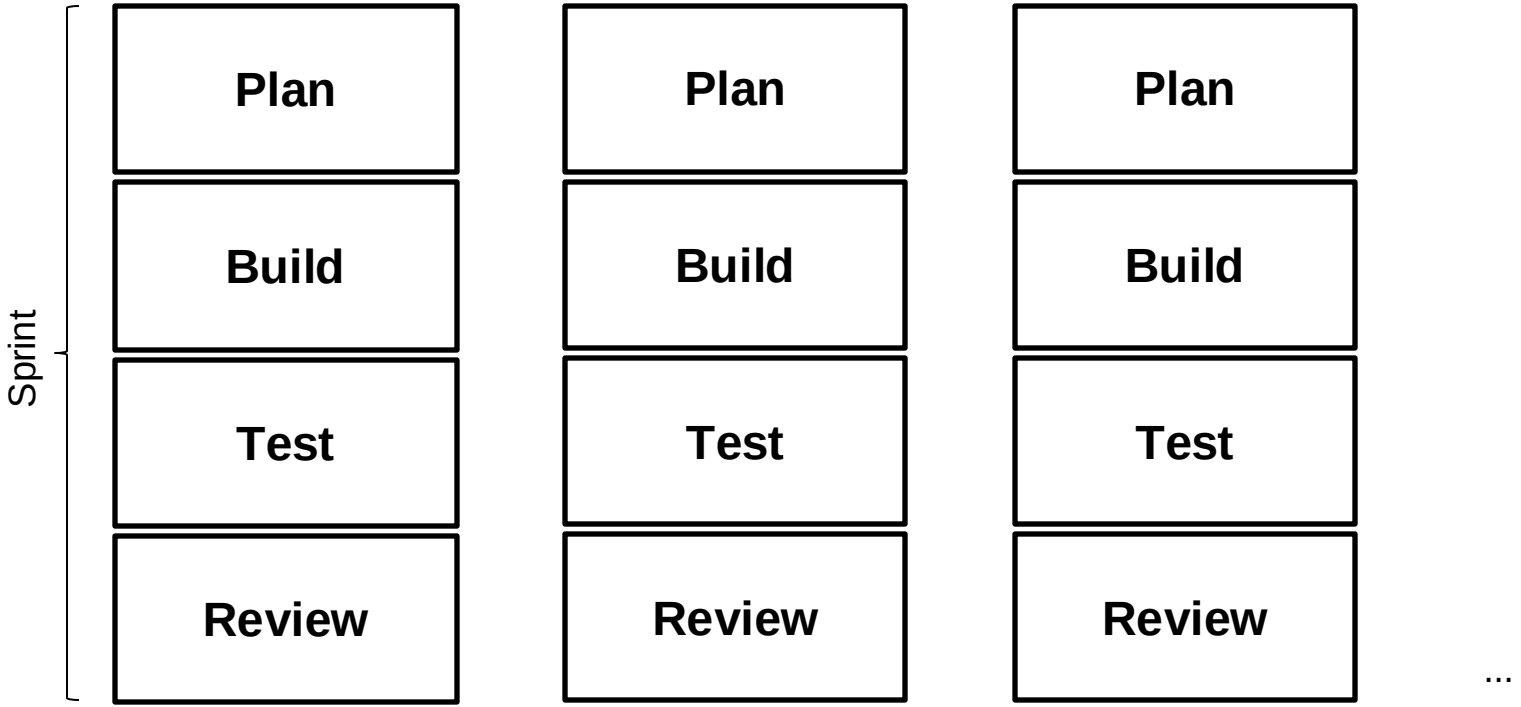
The 3 Key Ceremonies (Meetings)

Sprint Planning

Daily Scrum

Sprint Review

The Scrum Sprint Cycle



The Scrum Sprint Cycle

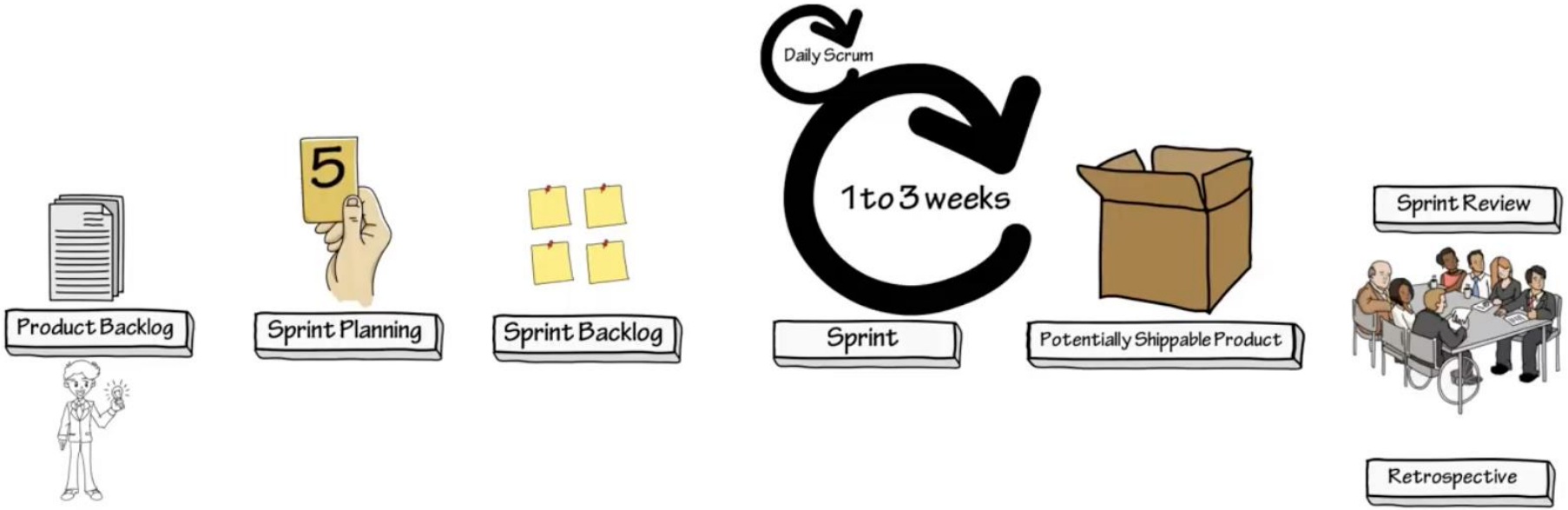
- Sprints are fixed length, normally 2–4 weeks
- The starting point for planning is the **product backlog**
  - i.e., the list of work to be done on the project
- The selection phase involves all of the project team who work with the customer to select the features and functionality from the product backlog to be developed during the sprint
  - Once these are agreed, the team members organize themselves to develop the software
- During development, the team is isolated from the customer and the organization, with all communications channelled through the so-called *Scrum master*
  - The role of the Scrum master is to protect the development team from external distractions
- At the end of the sprint, the work done is reviewed and presented to stakeholders. The next sprint cycle then begins.

Teamwork in Scrum

- The *Scrum master* is a facilitator who arranges daily meetings, tracks the backlog of work to be done, records decisions, measures progress against the backlog and communicates with customers and management outside of the team

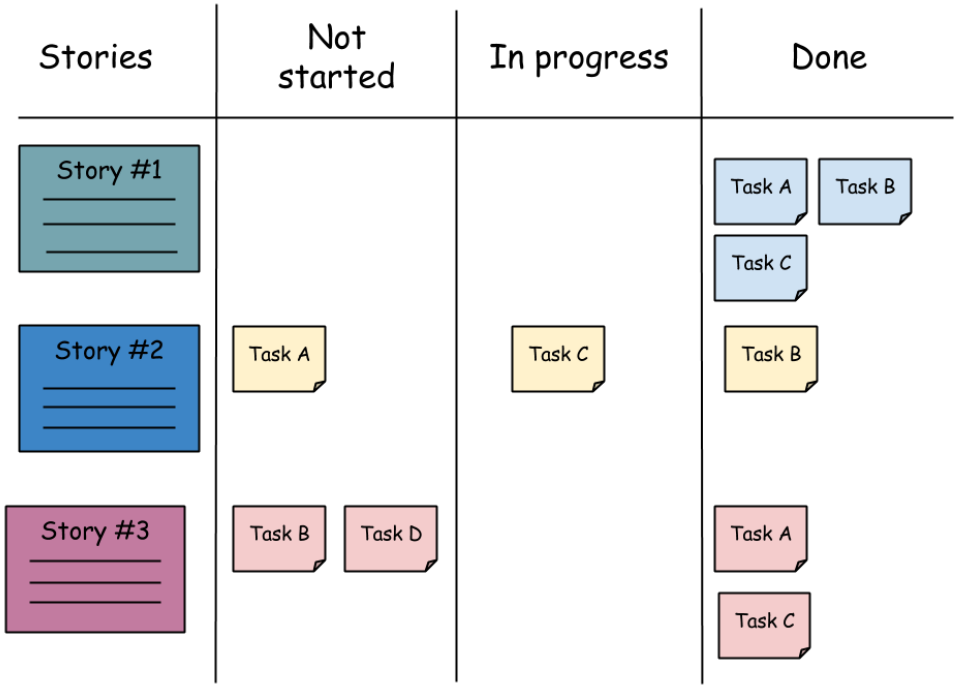
- The whole team attends short daily meetings (Scrums) where all team members share information, describe their progress since the last meeting, problems that have arisen and what is planned for the following day
  - This means that everyone on the team knows what is going on and, if problems arise, can re-plan short-term work to cope with them

Scrum in a Nutshell



Available tutorial at: https://www.youtube.com/watch?v=9TycLR0TqFA

Example of (an Agile) Scrum Board



Scrum Benefits

- The product is broken down into a set of manageable and understandable chunks
- Unstable requirements do not hold up progress
- The whole team has visibility of everything and consequently team communication is improved
- Customers see on-time delivery of increments and gain feedback on how the product works
- Trust between customers and developers is established and a positive culture is created in which everyone expects the project to succeed

Agile Project Management with Kanban

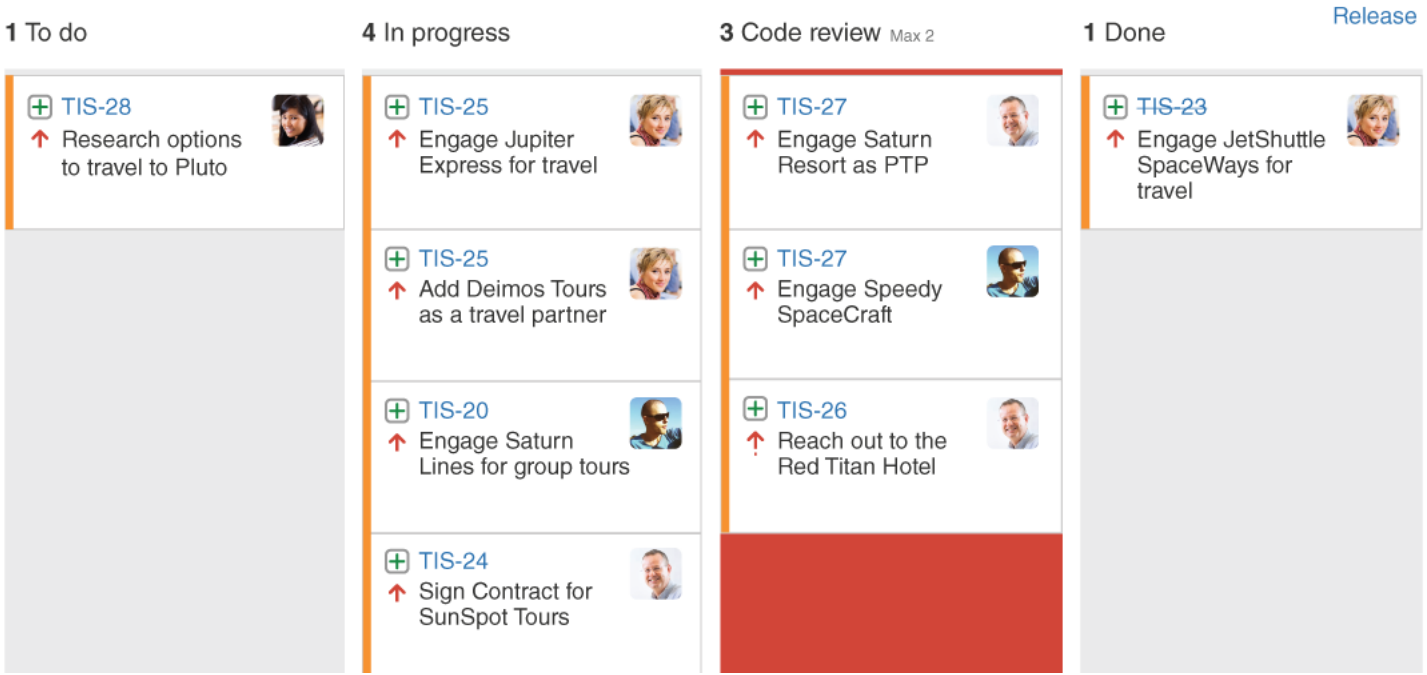
Kanban

- Similar to Scrum in terms of roles and ceremonies, just using different names
  - Scrum master → Agile Coach
  - Daily Scrum → Daily Stand-up meetings
  - Sprint Review → Demo
  - Retrospective

- Main differences
  - Kanban has no notion of *sprint*
    - Software development is a continuous process
  - Kanban limits the number of work-in-progress items/features in each column of the board

Kanban / Agile Board

- **Main difference with Scrum:** max number of items can be set in each column!



Remarks on Agile Project Management

- Both Scrum and Kanban are not as strict as it may appear from this presentation
  - they should be regarded as main guidelines rather than solid methodologies
  - can be changed based on the needs of the development team

- Best suited to small groups of developers required to implement software for which requirements may change
  - *This means when customers themselves have no clear idea of what they want!*

Scaling Agile Methods

Scaling Agile Methods

- Agile methods have proved to be successful for small and medium sized projects that can be developed by a small co-located team
- It is sometimes argued that the success of these methods comes because of improved communications which is possible when everyone is working together
- Scaling up agile methods involves changing these to cope with larger, longer projects where there are multiple development teams, perhaps working in different locations

Practical Problems with Agile Methods

- **Contractual Issues:** The informality of agile development is incompatible with the legal approach to contract definition that is commonly used in large companies

- **Software Maintenance:** Agile methods are most appropriate for new software development rather than software maintenance. Yet the majority of software costs in large companies come from maintaining their existing software systems

- Agile methods are **designed for small co-located teams,** yet much software development now involves worldwide distributed teams

Agile Principles and Organizational Practice

Principle	Practice
Customer involvement	This depends on having a customer who is willing and able to spend time with the development team and who can represent all system stakeholders. Often, customer representatives have other demands on their time and cannot play a full part in the software development. Where there are external stakeholders, such as regulators, it is difficult to represent their views to the agile team.
Embrace change	Prioritizing changes can be extremely difficult, especially in systems for which there are many stakeholders. Typically, each stakeholder gives different priorities to different changes.
Incremental delivery	Rapid iterations and short-term planning for development do not always fit in with the longer-term planning cycles of business planning and marketing. Marketing managers may need to know what product features several months in advance to prepare an effective marketing campaign.
Maintain simplicity	Under pressure from delivery schedules, team members may not have time to carry out desirable system simplifications.
People not process	Individual team members may not have suitable personalities for the intense involvement that is typical of agile methods, and therefore may not interact well with other team members.

Large System Development

- Large systems and their development processes are often constrained by external rules and regulations limiting the way that they can be developed
- Large systems have a long procurement and development time. It is difficult to maintain coherent teams who know about the system over that period as, inevitably, people move on to other jobs and projects
- Large systems usually have a diverse set of stakeholders. It is practically impossible to involve all of these different stakeholders in the development process

Scaling Up to Large Systems

- A completely incremental approach to requirements engineering is impossible
- There cannot be a single product owner or customer representative
- For large systems development, it is not possible to focus only on the code of the system
- Cross-team communication mechanisms have to be designed and used
- Continuous integration is practically impossible. However, it is essential to maintain frequent system builds and regular releases of the system

Multi-Team Scrum

- *Role replication*
  - Each team has a Product Owner for their work component and ScrumMaster
- *Product architects*
  - Each team chooses a product architect and these architects collaborate to design and evolve the overall system architecture
- *Release alignment*
  - The dates of product releases from each team are aligned so that a demonstrable and complete system is produced
- *Scrum of Scrums*
  - There is a daily Scrum of Scrums where representatives from each team meet to discuss progressand plan work to be done

Lessons Learned

- Agile methods are incremental development methods that focus on rapid software development, frequent releases of the software, reducing process overheads by minimizing documentation and producing high-quality code

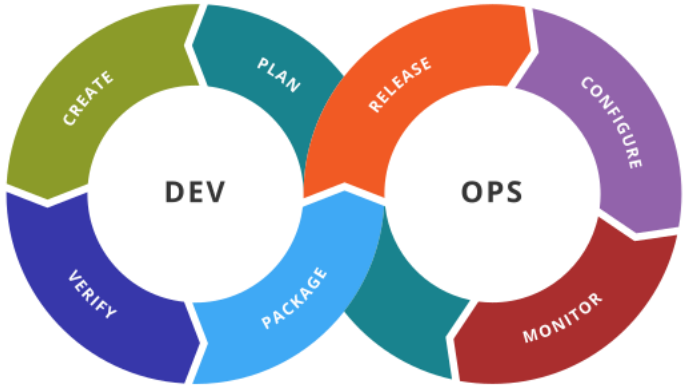
- Agile development practices include
  - User stories for system specification
  - Frequent releases of the software
  - Continuous software improvement
  - Test-first development
  - Customer participation in the development team

Lessons Learned

- Scrum and Kanban are agile methods that provide a project management framework
- Many practical development methods are a mixture of plan-based and agile development
- Scaling agile methods for large systems is difficult
  - Large systems need up-front design and some documentation and organizational practice may conflict with the informality of agile approaches

What's Next? DevOps = Development and Operations

1. **Plan**: software design and main features required
2. **Create**: code development and review, source code management tools, code merging, continuous integration tools, build status
3. **Verify**: continuous testing tools that provide feedback on business risks
4. **Package**: artifact repository, application pre-deployment staging
5. **Release**: change management, release approvals, release automation
6. **Configure**: infrastructure configuration and management, Infrastructure as Code tools
7. **Monitor**: applications performance monitoring, end–user experience



https://en.wikipedia.org/wiki/DevOps

What's Next? Continuous Delivery with Git

- Continuous delivery is an approach where teams release quality products frequently and predictably from source code repository to production in an automated fashion

Get Started

Continuous Integration

Continuous Deployment

CI vs CD vs CD

Plan

The business case for continuous delivery

Why choose Git for continuous delivery?

Is your team ready for devops?

Develop

Branching workflows for continuous delivery

Imitating "pure" continuous integration

CI-friendly Git repos

Test

The nuts n' bolts of continuous integration

Doing continuous integration on feature branches

Running tests in parallel

Deliver

Understanding when are you ready to ship

Deployment projects for fun and profit

Branch based deployments

https://www.atlassian.com/continuous-delivery

Further Readings

- DevOps and Continuous Delivery
  - https://www.atlassian.com/agile
  - https://www.atlassian.com/devops
  - https://aws.amazon.com/devops/
  - https://aws.amazon.com/devops/continuous-integration/
  - https://about.gitlab.com/devops/
  - https://continuousdelivery.com/

- DevSecOps (security by design with integrated continuous security testing)
  - https://about.gitlab.com/solutions/dev-sec-ops/