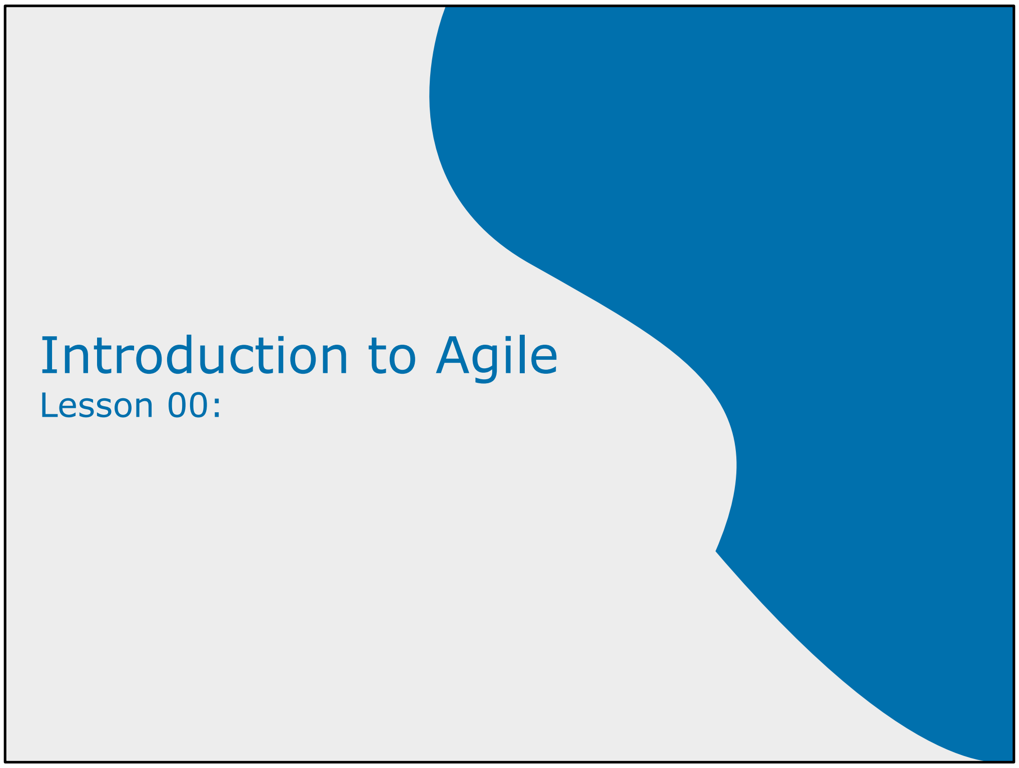


Introduction to Agile

Lesson 00:

Course Goals and Non Goals

Course Goals

- Introducing participants to the Agile Software Development Model
- Introduction to Agile Practices & Methods
- Understanding SCRUM
- At the end of this program, participants gain an understanding of how to transition sound traditional test practices into an Agile Development Environment
- Understand the key differences between traditional and Agile Testing Practices
- Understand the roles and responsibilities of a typical Agile Testing Team

Course Non Goals

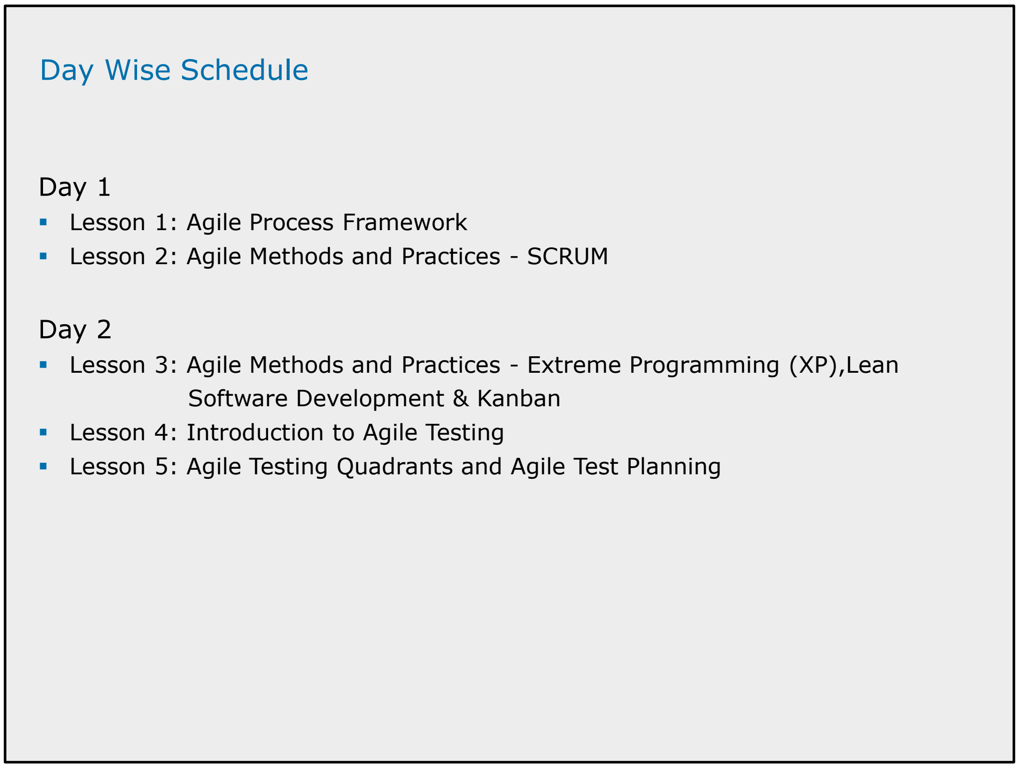
- This course does not cover other than anything the course goals

Pre-requisites

Basic knowledge of Software Development Life Cycle
Basic knowledge of Programming Concepts
Basic Knowledge of Software Testing Fundamentals

Intended Audience

Novice Developers
Test Engineers



Day Wise Schedule

Day 1

- Lesson 1: Agile Process Framework
- Lesson 2: Agile Methods and Practices - SCRUM

Day 2

- Lesson 3: Agile Methods and Practices - Extreme Programming (XP),Lean Software Development & Kanban
- Lesson 4: Introduction to Agile Testing
- Lesson 5: Agile Testing Quadrants and Agile Test Planning

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

References

Websites :

- www.extremeprogramming.org
- www.agilemanifesto.org
- www.wikipedia.org
- www.scrum-institute.org
- www.agilealliance.org
- www.agilemodeling.com
- www.scrumguides.org
- www.mountaingoatsoftware.com

Books :

- Agile Testing: A Practical Guide for Testers and Agile Teams - Lisa Crispin, Janet Gregory
- User Stories Applied: For Agile Software Development – Mike Cohn

Other Parallel Technology Areas

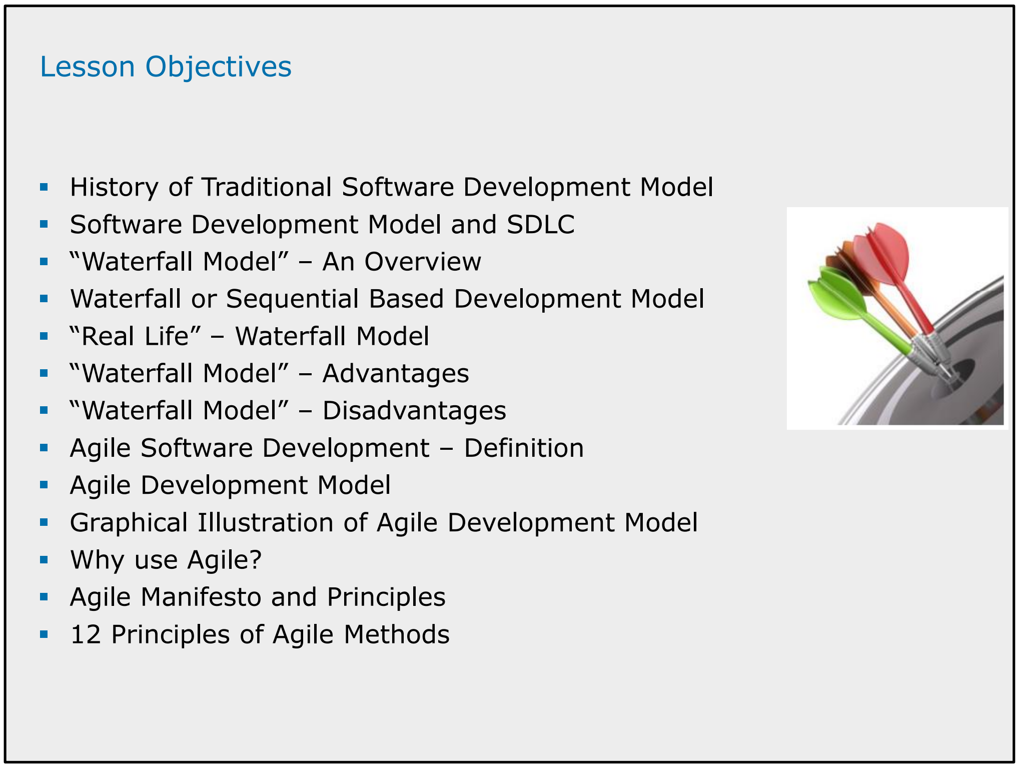
None

Next Step Courses

None

Introduction to Agile

Lesson 1: Agile Process Framework



Lesson Objectives

- History of Traditional Software Development Model
- Software Development Model and SDLC
- “Waterfall Model” – An Overview
- Waterfall or Sequential Based Development Model
- “Real Life” – Waterfall Model
- “Waterfall Model” – Advantages
- “Waterfall Model” – Disadvantages
- Agile Software Development – Definition
- Agile Development Model
- Graphical Illustration of Agile Development Model
- Why use Agile?
- Agile Manifesto and Principles
- 12 Principles of Agile Methods

Lesson Objectives

- Agile Values
- What is NOT an Agile software development?
- Foundation of an Agile software development Method
- Common Characteristics of Agile Methods
- Agile Methods and Practices
- When to use Agile Model?
- Advantages of Agile Model
- Disadvantages of Agile Model
- Difference between Agile and Waterfall Model
- Agile – Myths and Reality
- Agile Market Insight

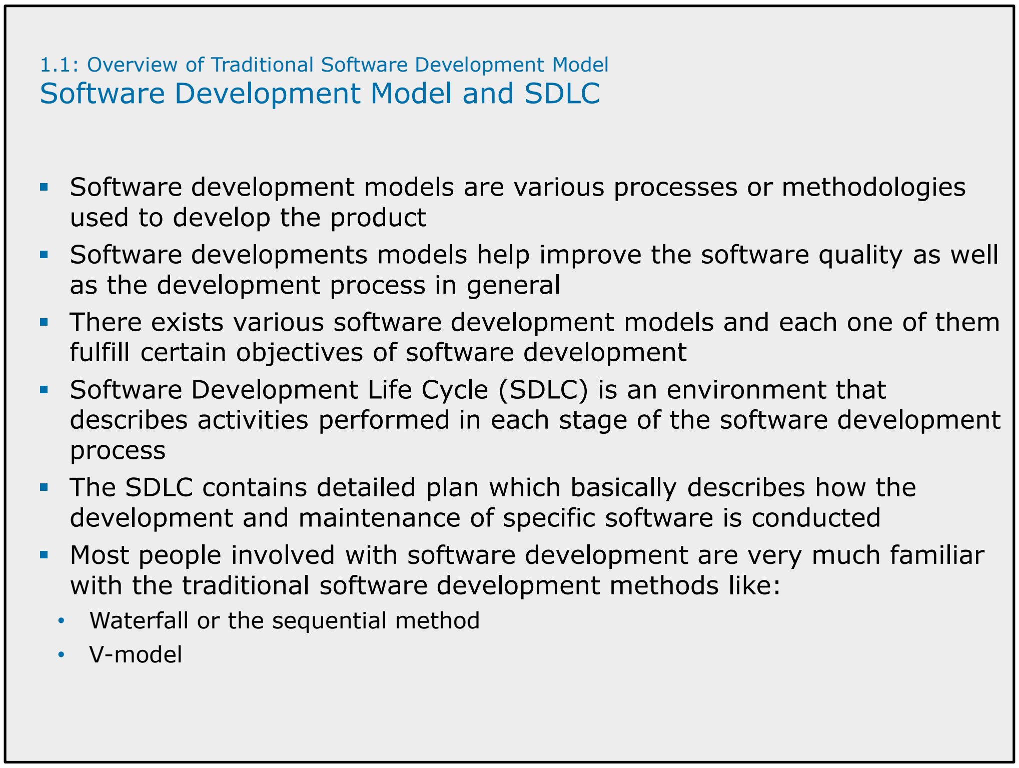
1.1: Overview of Traditional Software Development Model

History of Traditional Software Development Model

- Traditional software development methodologies are often called heavyweight methodologies as they are based on a sequential series of steps that has to be defined and documented in detail
- These methodologies are based on progressive series of steps like requirements definition, design and architectural planning, development and testing
- The traditional software development models depends upon a set of predefined processes
- The success of a project which is build with traditional software development model depends upon how well the requirements are stated before the project begins
- In this approach implementing changes during development life cycle is somewhat critical
- However, this approach also poses benefits like easy estimation of cost of the project, scheduling and allocation of resources

History of Traditional Software Development Model

The traditional software development models like Waterfall Model, V-Model and RUP are classified into the heavyweight methodologies. These methodologies are based on progressive series of steps like requirements definition, design and architectural planning, development and testing. Traditional software development methodologies require defining and documenting a stable set of requirements at the beginning of a project.
The traditional software development model comprises of four basic phases:
The first phase is to set up the system requirements for the project and defining the amount of time it will take to implement the various phases of development life cycle.
Once the requirements are finalized, the next step moves into the design and architectural planning phase. In this phase the technical infrastructure is created in the form of diagrams or models.
Once the team is satisfied with the architectural and design plan, the project moves into the development phase where code is produced until the specific goals are reached. Development is often broken down into smaller tasks that are distributed among various teams based on skill.
The testing phase often overlaps with the development phase to ensure issues are addressed early on. Once the project nears completion and the developers are close to meeting the project requirements, the customer will become part of the testing and feedback cycle and the project was delivered after the customer satisfy with it.
The success of a project which is build with traditional software development model depends upon how well the requirements are stated before the project begins. In this approach implementing changes during development life cycle is somewhat critical. However, this approach also poses benefits like easy estimation of cost of the project, scheduling and allocation of resources.



1.1: Overview of Traditional Software Development Model

Software Development Model and SDLC

- Software development models are various processes or methodologies used to develop the product
- Software developments models help improve the software quality as well as the development process in general
- There exists various software development models and each one of them fulfill certain objectives of software development
- Software Development Life Cycle (SDLC) is an environment that describes activities performed in each stage of the software development process
- The SDLC contains detailed plan which basically describes how the development and maintenance of specific software is conducted
- Most people involved with software development are very much familiar with the traditional software development methods like:
  - Waterfall or the sequential method
  - V-model

Software Development Model and SDLC

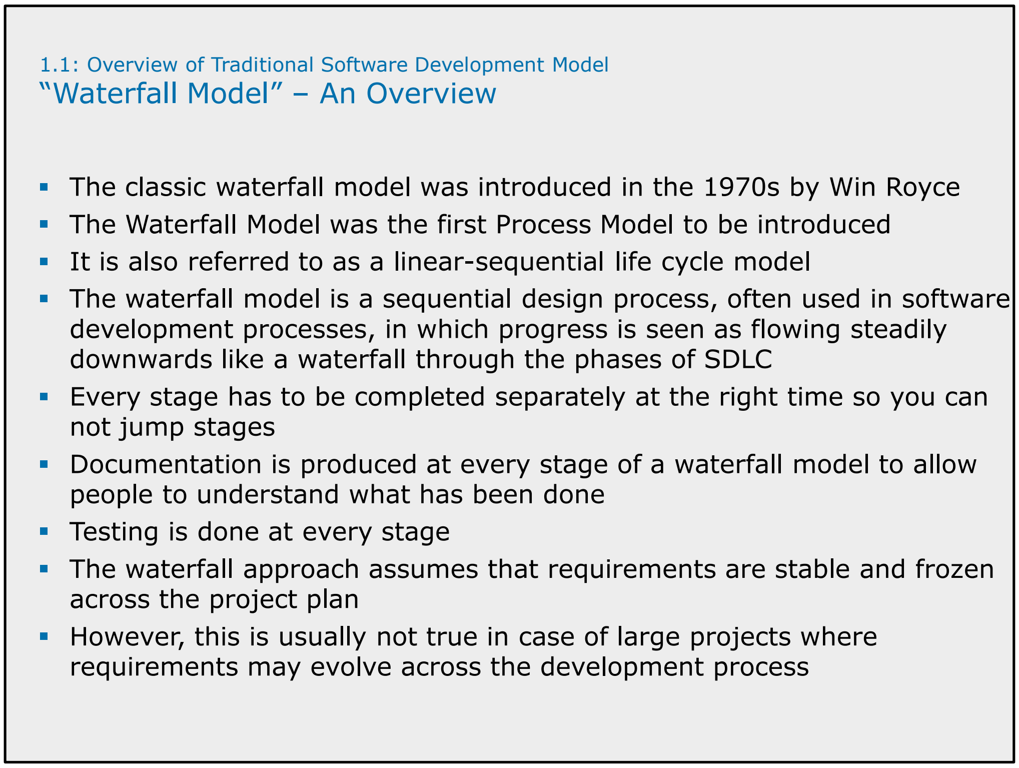
A software application or an information system is designed to perform a particular set of tasks. Often, this set of tasks that the system will perform provides well-defined results, which involve complex computation and processing. Therefore it's a very important and tedious job to administer the entire development process to ensure that the end product comprises of high degree of integrity and robustness, as well as user acceptance.

Software Development Life Cycle (SDLC) is a process of building or maintaining software systems. The SDLC process primarily takes care of various pre-development phases like requirement gathering, requirement analysis, design and architecture as well as post development activities like testing and validation. It also consists of the models and methodologies that development teams use to develop the software systems, which the methodologies form the framework for planning and controlling the entire development process.

Currently, there are two SDLC methodologies which are utilized by most system developers, namely the traditional development and agile development.

Most people involved with software development are very much familiar with the traditional software development methods like:

1. Waterfall or the sequential method
2. V-model



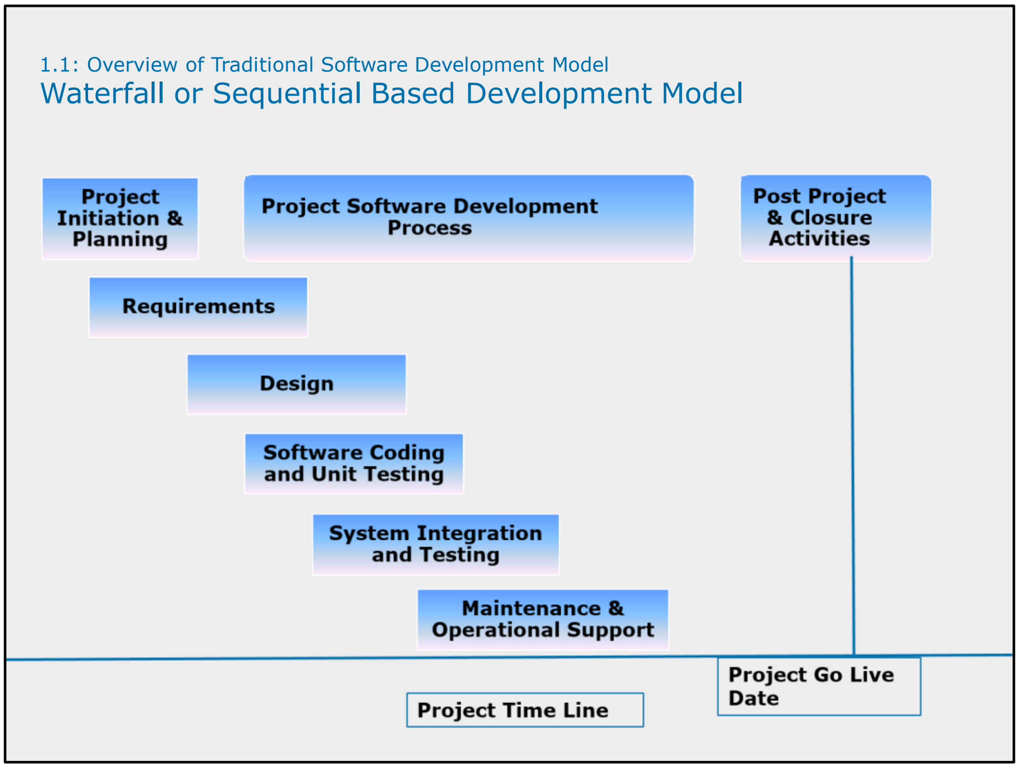
1.1: Overview of Traditional Software Development Model

“Waterfall Model” – An Overview

- The classic waterfall model was introduced in the 1970s by Win Royce
- The Waterfall Model was the first Process Model to be introduced
- It is also referred to as a linear-sequential life cycle model
- The waterfall model is a sequential design process, often used in software development processes, in which progress is seen as flowing steadily downwards like a waterfall through the phases of SDLC
- Every stage has to be completed separately at the right time so you can not jump stages
- Documentation is produced at every stage of a waterfall model to allow people to understand what has been done
- Testing is done at every stage
- The waterfall approach assumes that requirements are stable and frozen across the project plan
- However, this is usually not true in case of large projects where requirements may evolve across the development process

Waterfall or Sequential Based Development Model

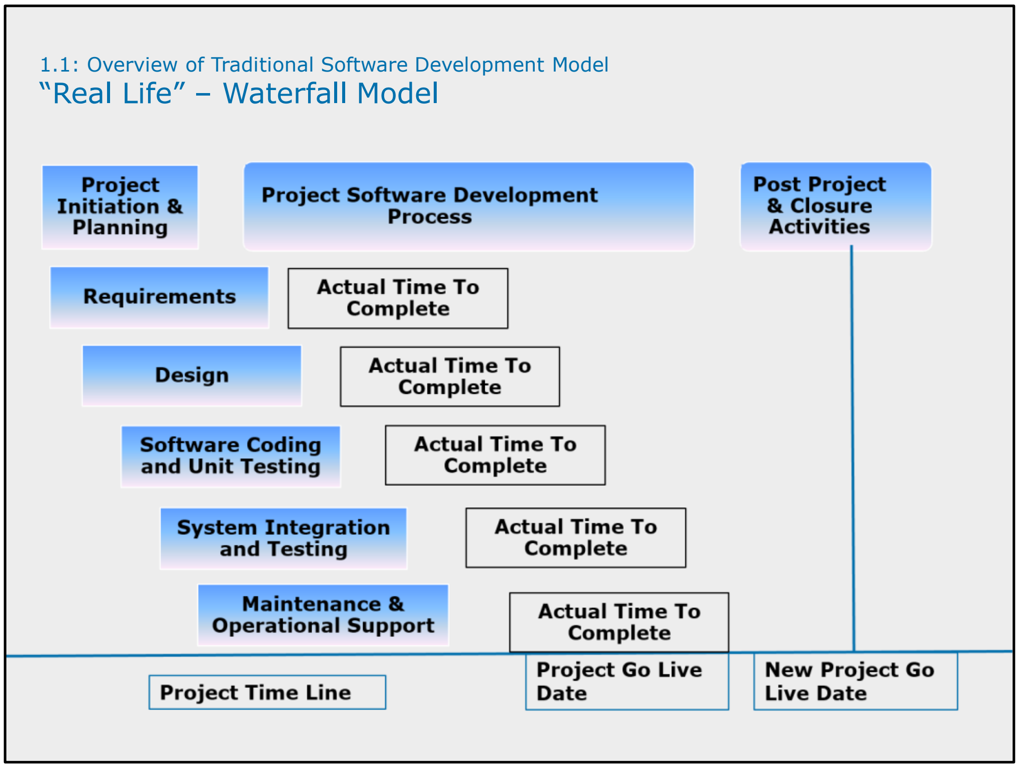
The waterfall model was very simple and logical in the way it approached software development and for these reasons alone it is still in use today. It takes very little time for anybody new to a development team or project to understand a sequential based process.
While it may be logical, one of its biggest problems was ensuring that the time spent in each phase didn't overrun into the next one resulting in a series of compounding delays. Unfortunately, this was the case more times than not and ultimately the testing phase got "squeezed" against the deadline, which became an immovable date.
Or if the project timelines are fixed then either parts of the original scope and functionality may not be delivered or even worse the functionality is delivered, but there is no time for sufficient testing and the quality of the product suffers.



1.1: Overview of Traditional Software Development Model

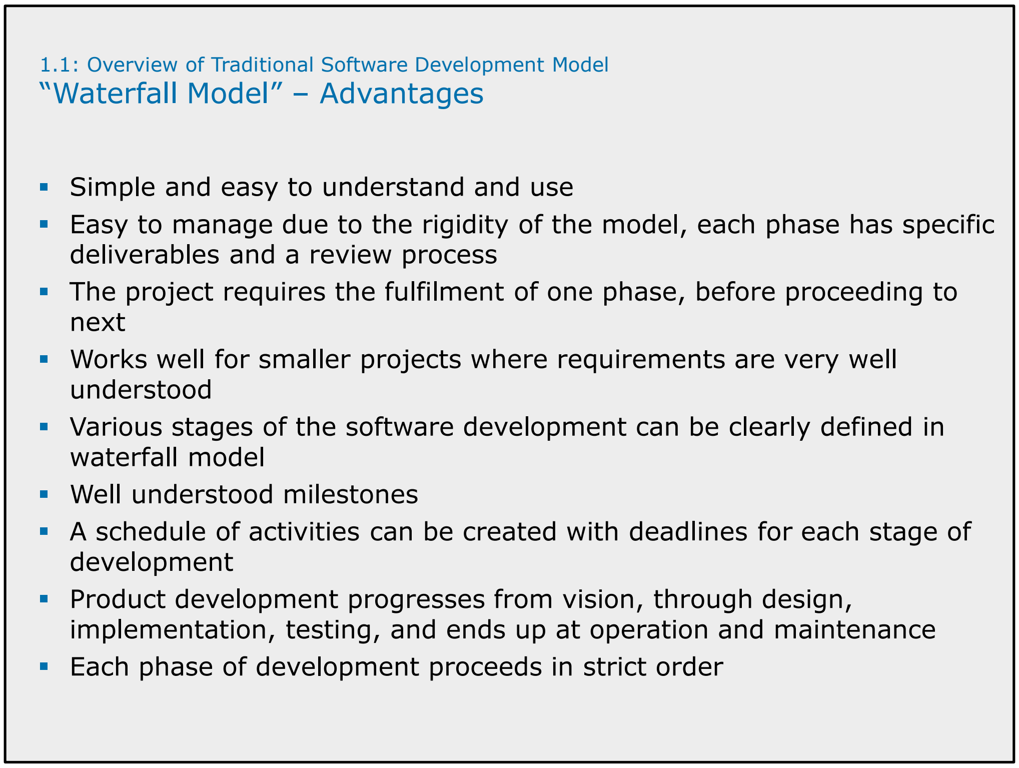
Waterfall or Sequential Based Development Model

Project Initiation & Planning
Project Software Development Process
Post Project & Closure Activities
Requirements
Design
Software Coding and Unit Testing
System Integration and Testing
Maintenance & Operational Support
Project Go Live Date
Project Time Line



1.1: Overview of Traditional Software Development Model

"Real Life" – Waterfall Model



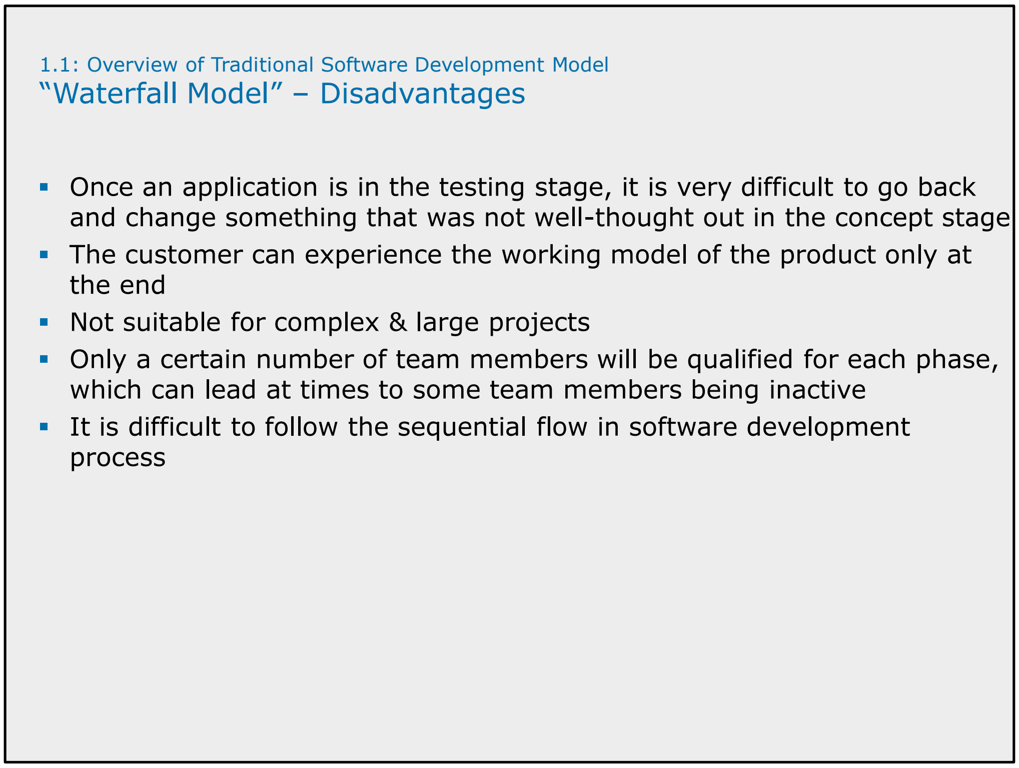
1.1: Overview of Traditional Software Development Model

“Waterfall Model” – Advantages

- Simple and easy to understand and use
- Easy to manage due to the rigidity of the model, each phase has specific deliverables and a review process
- The project requires the fulfilment of one phase, before proceeding to next
- Works well for smaller projects where requirements are very well understood
- Various stages of the software development can be clearly defined in waterfall model
- Well understood milestones
- A schedule of activities can be created with deadlines for each stage of development
- Product development progresses from vision, through design, implementation, testing, and ends up at operation and maintenance
- Each phase of development proceeds in strict order

“Waterfall Model” – Advantages

Waterfall model is simple to implement and also the amount of resources required for it are minimal. In this model, output is generated after each stage (as seen before), therefore it has high visibility. The client and project manager gets a feel that there is considerable progress. Here it is important to note that in any project psychological factors also play an important role.
Project management, both at internal level and client's level, is easy again because of visible outputs after each phase. Deadlines can be set for the completion of each phase and evaluation can be done from time to time, to check if project is going as per milestones.
This methodology is significantly better than the haphazard approach to develop software. It provides a template into which methods of analysis, design, coding, testing and maintenance can be placed.
This methodology is preferred in projects where quality is more important as compared to schedule or cost.



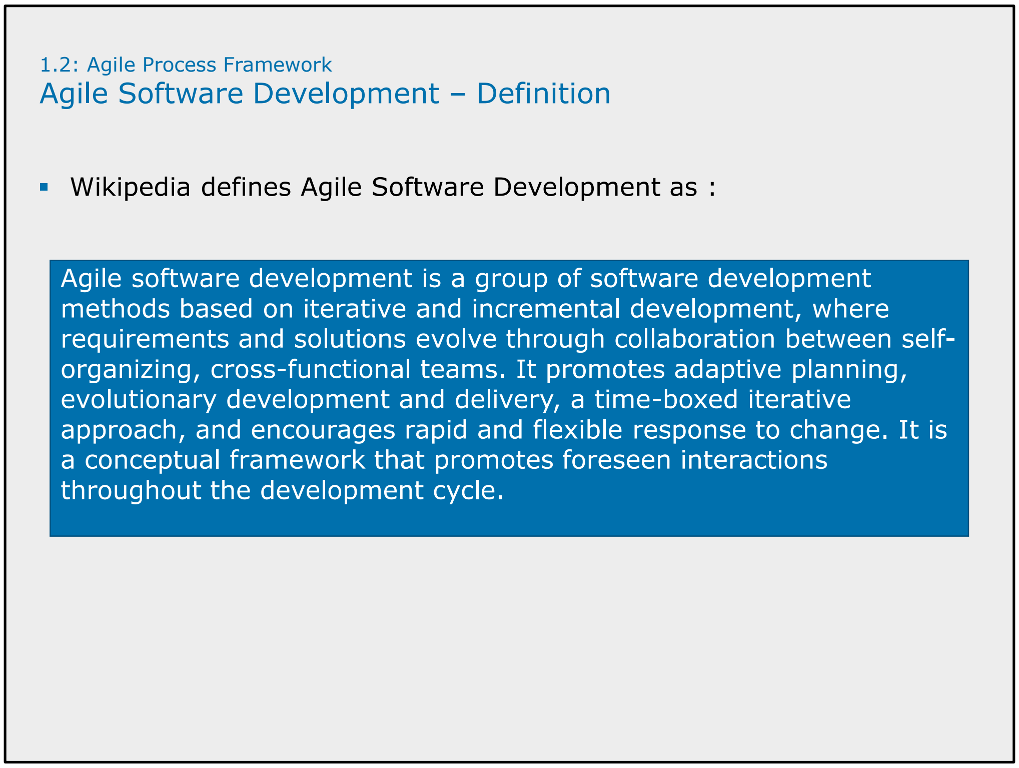
1.1: Overview of Traditional Software Development Model

“Waterfall Model” – Disadvantages

- Once an application is in the testing stage, it is very difficult to go back and change something that was not well-thought out in the concept stage
- The customer can experience the working model of the product only at the end
- Not suitable for complex & large projects
- Only a certain number of team members will be qualified for each phase, which can lead at times to some team members being inactive
- It is difficult to follow the sequential flow in software development process

“Waterfall Model” – Disadvantages

The Waterfall model exhibits poor flexibility. The majority of software is written as part of a contract with a client, and clients are notorious for changing their stated requirements. Thus the software project must be adaptable, and spending considerable effort in design and implementation based on the idea that requirements will never change is neither adaptable nor realistic in these cases.
The waterfall model however is argued by many to be a bad idea in practice, mainly because of their belief that it is impossible to get one phase of a software product's lifecycle "perfected" before moving on to the next phases and learning from them. A typical problem is when requirements change midway through, resulting in a lot of time and effort wastage due to "Big Design Up Front".
Constant testing from the design, implementation and verification phases is required to validate the phases preceding them. Users of the waterfall model may argue that if designers follow a disciplined process and do not make mistakes that there is no need to constantly validate the preceding phases.



1.2: Agile Process Framework

Agile Software Development – Definition

- Wikipedia defines Agile Software Development as :

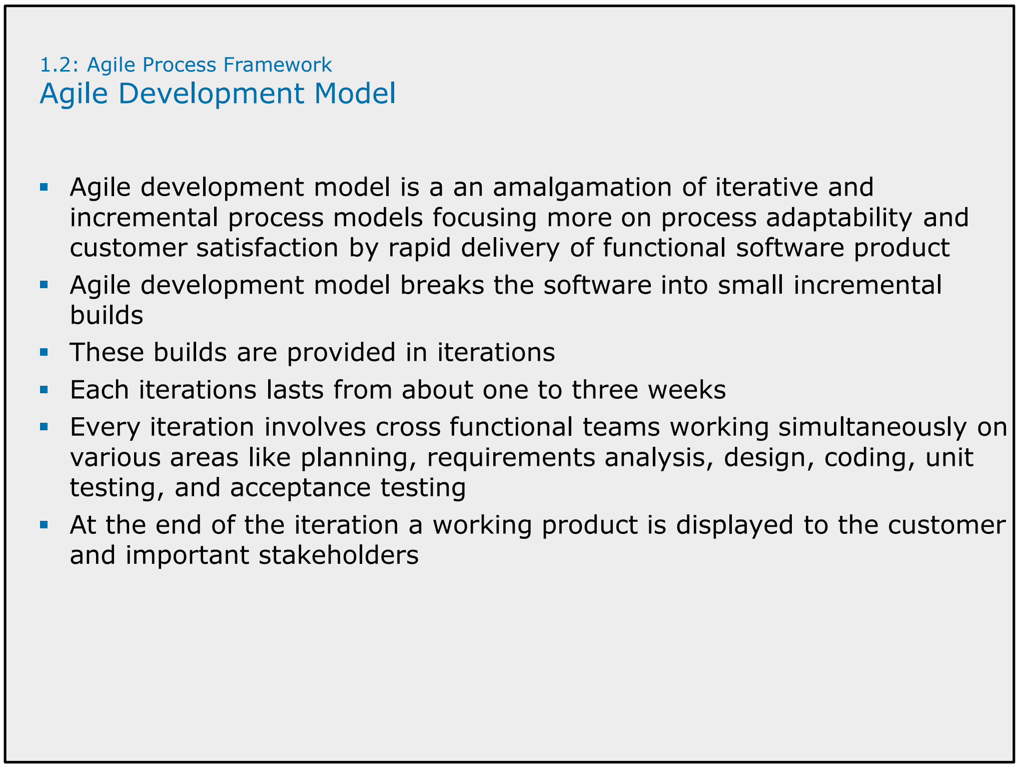
Agile software development is a group of software development methods based on iterative and incremental development, where requirements and solutions evolve through collaboration between self-organizing, cross-functional teams. It promotes adaptive planning, evolutionary development and delivery, a time-boxed iterative approach, and encourages rapid and flexible response to change. It is a conceptual framework that promotes foreseen interactions throughout the development cycle.

1.2: Agile Process Framework

Agile Software Development – Definition

- The web site, SearchSoftwareQuality.com contains the following definition:

In software application development, agile software development (ASD) is a methodology for the creative process that anticipates the need for flexibility and applies a level of pragmatism into the delivery of the finished product. Agile software development focuses on keeping code simple, testing often, and delivering functional bits of the application as soon as they're ready. The goal of ASD is to build upon small client-approved parts as the project progresses, as opposed to delivering one large application at the end of the project.



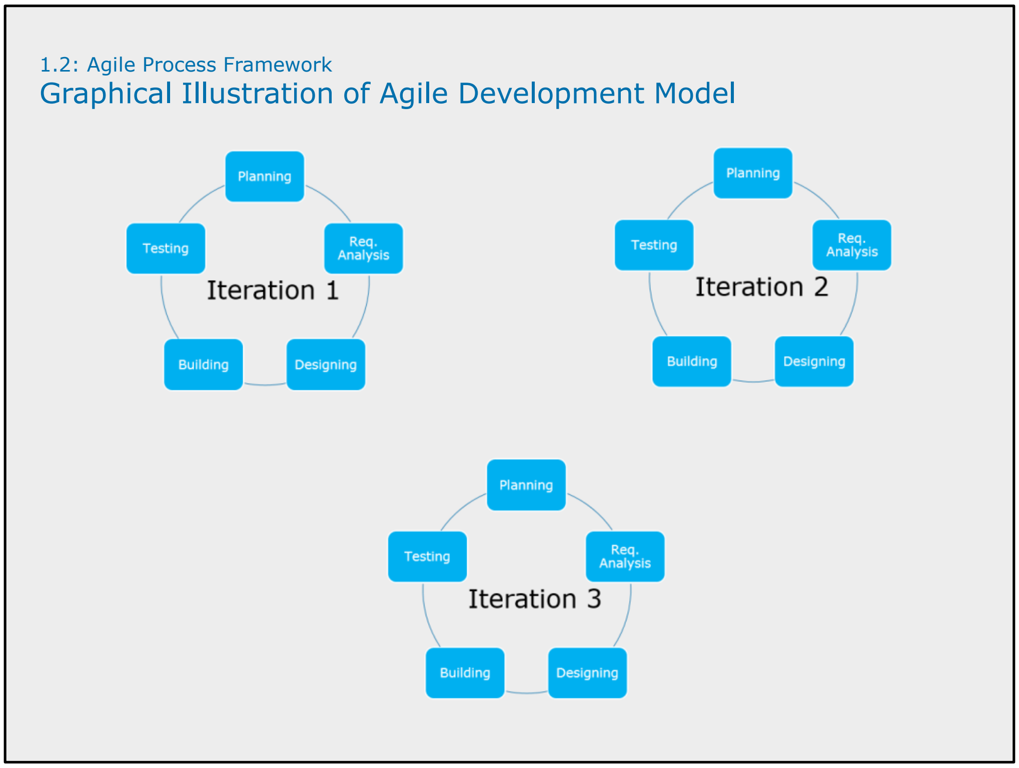
1.2: Agile Process Framework

Agile Development Model

- Agile development model is a an amalgamation of iterative and incremental process models focusing more on process adaptability and customer satisfaction by rapid delivery of functional software product
- Agile development model breaks the software into small incremental builds
- These builds are provided in iterations
- Each iterations lasts from about one to three weeks
- Every iteration involves cross functional teams working simultaneously on various areas like planning, requirements analysis, design, coding, unit testing, and acceptance testing
- At the end of the iteration a working product is displayed to the customer and important stakeholders

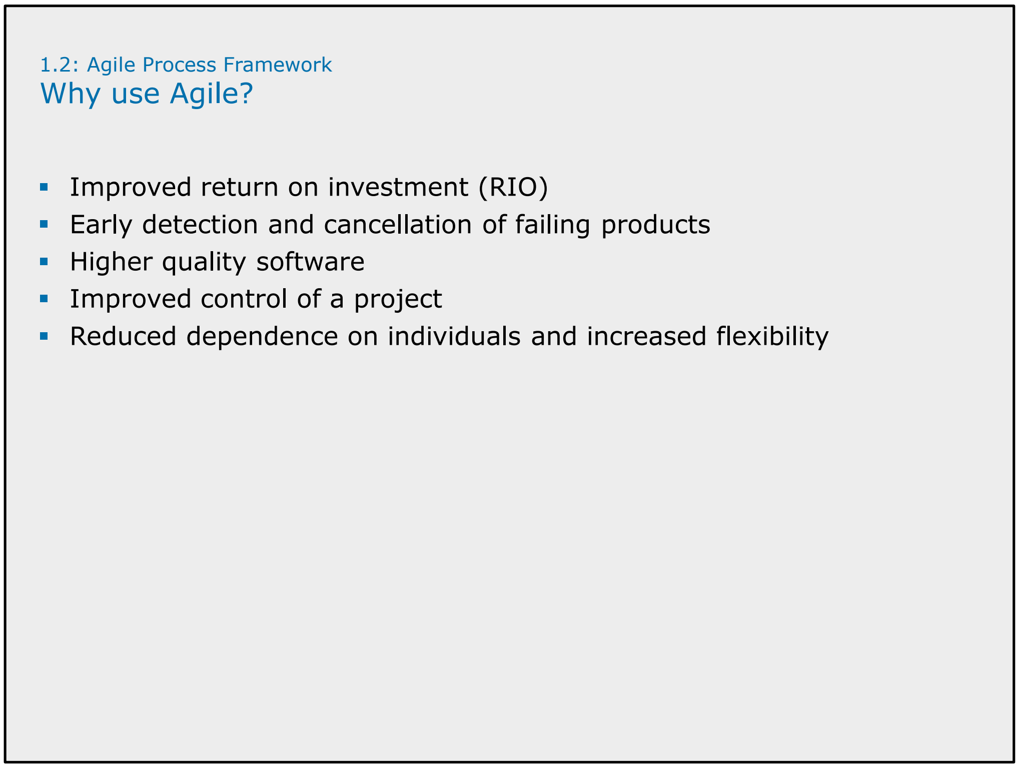
Agile Development Model

Agile Development Model works on the basis of modern development approach which strongly believes that every project and every project needs are different. To attain the project success one needs to make an effort to tailored the existing methodologies to best suit the changing project requirements. In agile the tasks are divided to time boxes (small time frames) to deliver specific features for a release. Iterative approach is taken and working software build is delivered after each iteration. Each build is incremental in terms of features; the final build holds all the features required by the customer.
Agile thought process had started early in the software development and started becoming popular with time due to its flexibility and adaptability. The most popular agile methods include Rational Unified Process (1994), Scrum (1995), Crystal Clear, Extreme Programming (1996), Adaptive Software Development, Feature Driven Development, and Dynamic Systems Development Method (DSDM) (1995). These are now collectively referred to as agile methodologies, after the Agile Manifesto was published in 2001.



1.2: Agile Process Framework

Graphical Illustration of Agile Development Model



1.2: Agile Process Framework

Why use Agile?

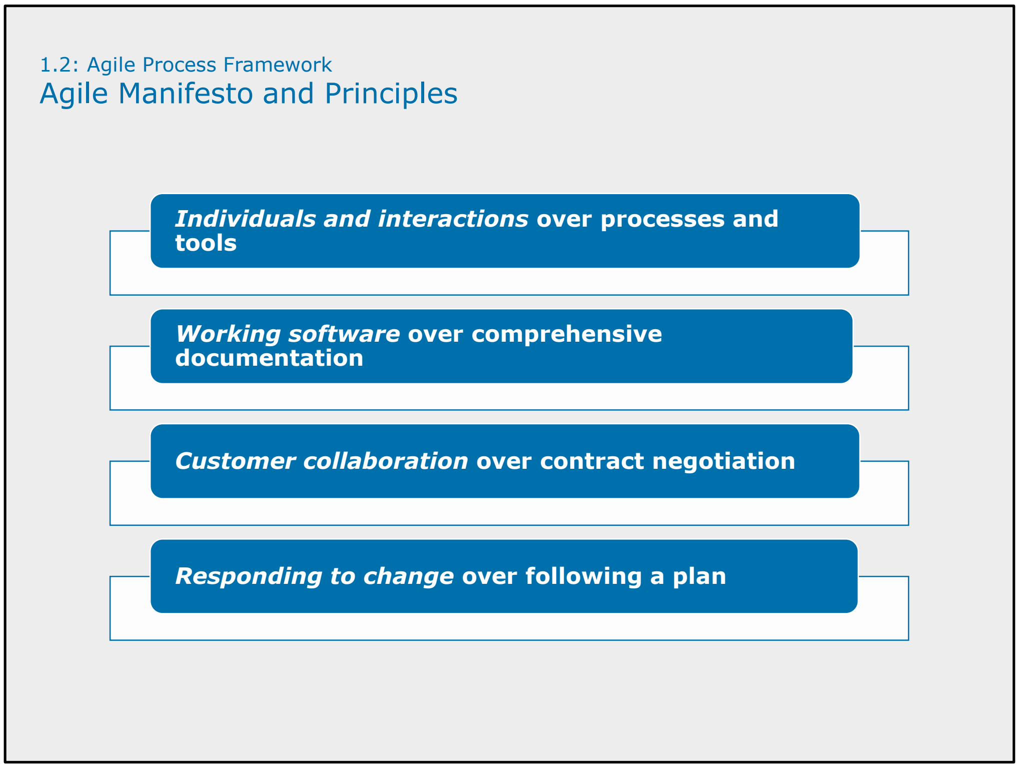
- Improved return on investment (RIO)
- Early detection and cancellation of failing products
- Higher quality software
- Improved control of a project
- Reduced dependence on individuals and increased flexibility

Why use Agile?

Improved return on investment: This is the fundamental reason to use agile methods, and it's achieved in a number of different ways. In an agile project, the initial requirements are the baseline for ROI. If the project runs to completion with no changes, then the business will get the projected returns. However, by providing frequent opportunities for customer feedback, agile methods let customers steer the project incrementally, taking advantage of new insights or changed circumstances to build a better system and improve the ROI. By delivering working software early and often, agile projects also present opportunities for early deployment and provide earlier return on smaller initial investments.

Early cancellation of failing projects: It's common to observe projects to be on track for the first 80% or more of the schedule, only to be delayed for many months. In this situation sponsors face a difficult choice. Do they abandon the project after spending 80% of the budget or continue to fund the project in the hope of getting some return on the investment?

Higher quality: Of the four fundamental variables you can use to control a software development project, cost, time, scope and quality, most agile methods explicitly use scope as their control variable. All agile methods emphasize the production of high quality software, and extreme programming in particular adds a number of practices to support this objective.



1.2: Agile Process Framework

Agile Manifesto and Principles

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

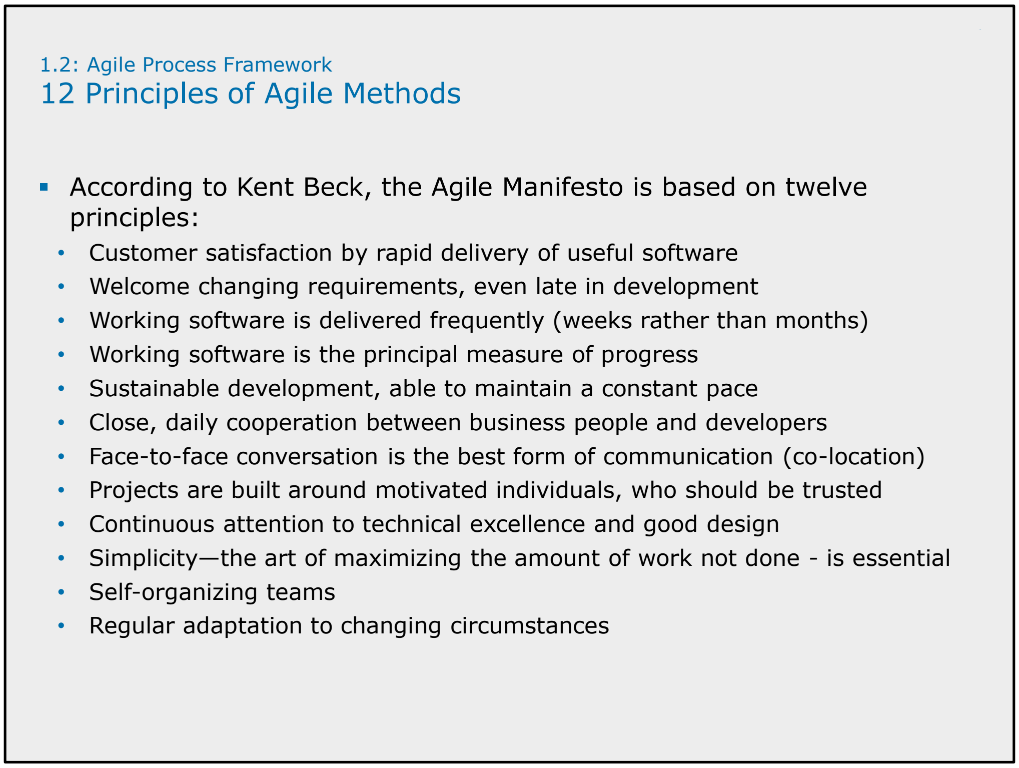
Agile Manifesto and Principles

The agile manifesto is the result of a meeting at the Snowbird ski resort in Utah in 2001. Prior to that date, the individual agile processes were referred to as lightweight.
The release of the manifesto immediately gave the industry a tangible definition of agile and ground rules for adding new ideas in the future. The manifesto provides clear direction and is used to discuss and compare agile methodologies. The manifesto provides one common roof for all agilists, whatever their favorite agile methodology might be.

Here are the core values of the manifesto:

1. **Individuals and interaction take precedence over processes and tools.**
2. **Working software takes precedence over comprehensive documentation.**
3. **Customer collaboration takes precedence over contract negotiation.**
4. **Responding to change takes precedence over following a plan.**

Please note that the left side of each statement is valued more than the right side. What is important and often misunderstood is that the manifesto does not recommend neglecting the values of the right side - for example, project documentation. It simply means that the values on the left are valued more highly. Every agile project team has to find the right balance as a team, but also they must find balance within the organization.



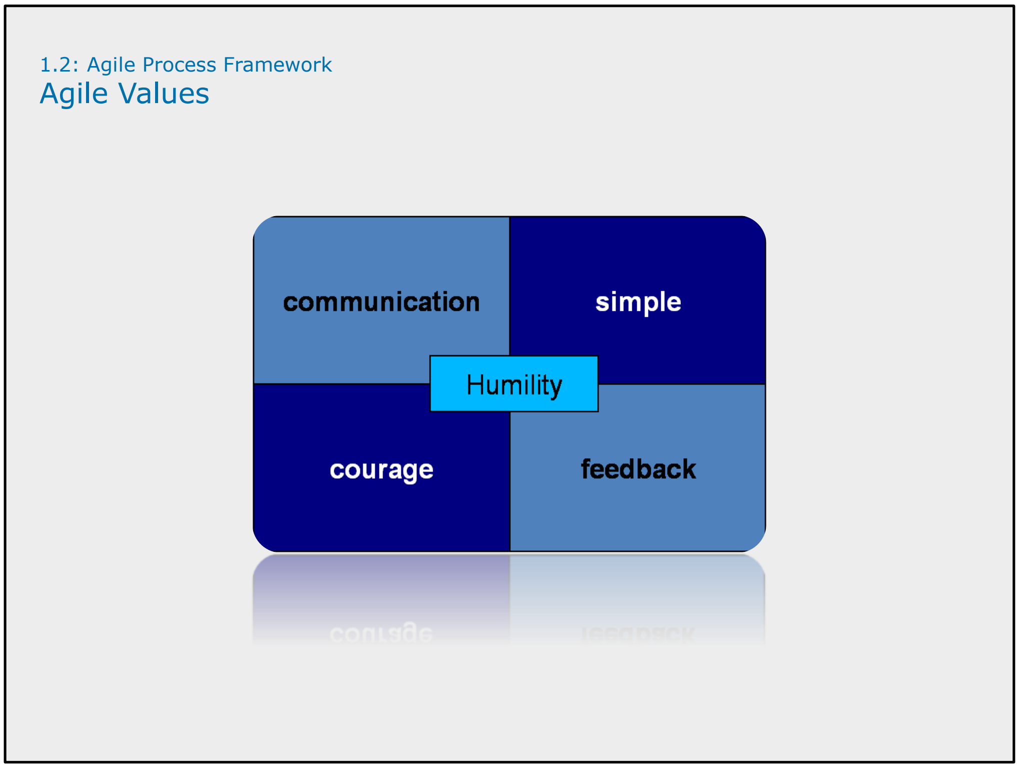
1.2: Agile Process Framework

12 Principles of Agile Methods

- According to Kent Beck, the Agile Manifesto is based on twelve principles:
  - Customer satisfaction by rapid delivery of useful software
  - Welcome changing requirements, even late in development
  - Working software is delivered frequently (weeks rather than months)
  - Working software is the principal measure of progress
  - Sustainable development, able to maintain a constant pace
  - Close, daily cooperation between business people and developers
  - Face-to-face conversation is the best form of communication (co-location)
  - Projects are built around motivated individuals, who should be trusted
  - Continuous attention to technical excellence and good design
  - Simplicity—the art of maximizing the amount of work not done - is essential
  - Self-organizing teams
  - Regular adaptation to changing circumstances

The Principles of Agile Methods – www.agilemanifesto.org

1. Our highest priority is to satisfy the customer through the early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity--the art of maximizing the amount of work not done--is essential.
11. The best architectures, requirements, and designs emerge from self-organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.



1.2: Agile Process Framework

Agile Values

The five values of Agile Modelling (AM) are:

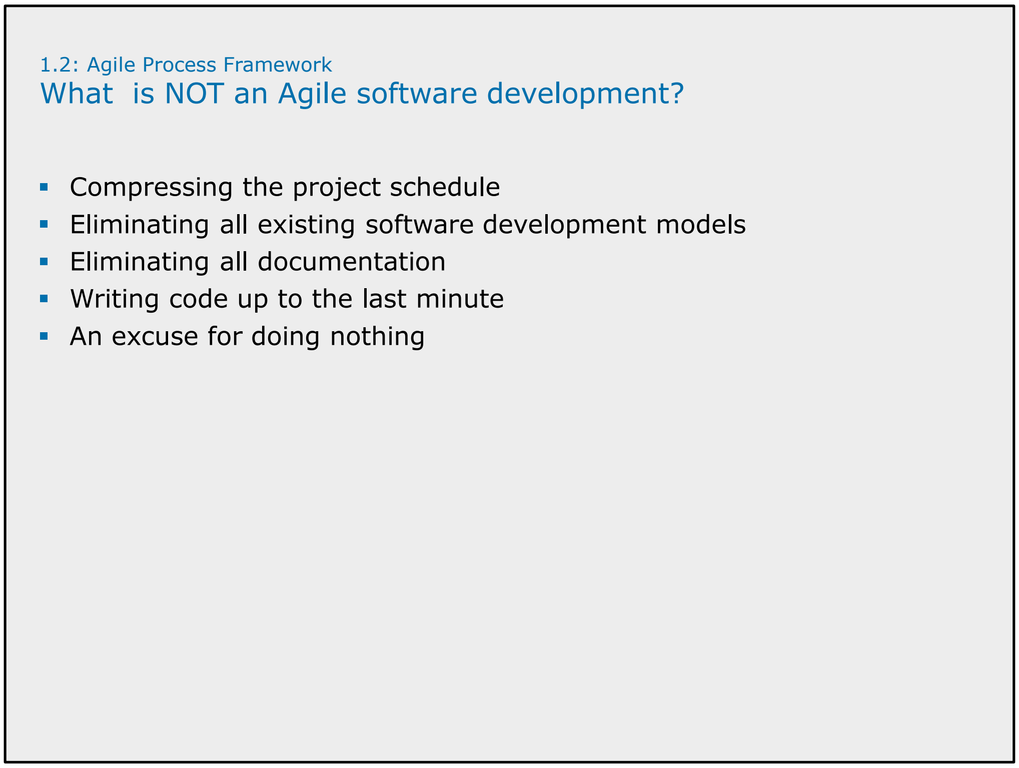
1. Communication: Models promote communication between your team and your project stakeholders as well as between developers on your team.

2. Simplicity: It's important that developers understand that models are critical for simplifying both software and the software process, it's much easier to explore an idea, and improve upon it as your understanding increases, by drawing a diagram or two instead of writing tens or even hundreds of lines of code.

3. Feedback: Kent Beck says it best in Extreme Programming Explained "Optimism is an occupational hazard of programming, feedback is the treatment." By communicating your ideas through diagrams, you quickly gain feedback, enabling you to act on that advice.

4. Courage: Courage is important because you need to make important decisions and be able to change direction by either discarding or refactoring your work when some of your decisions prove inadequate.

5. Humility - The best developers have the humility to recognize that they don't know everything, that their fellow developers, their customers, and in fact all project stakeholders also have their own areas of expertise and have value to add to a project. An effective approach is to assume that everyone involved with your project has equal value and therefore should be treated with respect. Huet Landry suggests the concept of "Other Esteem", instead of "Self Esteem", where you treat the opinions of others as if they have more value than yours. With this approach your first reaction to another's idea will be most positive.



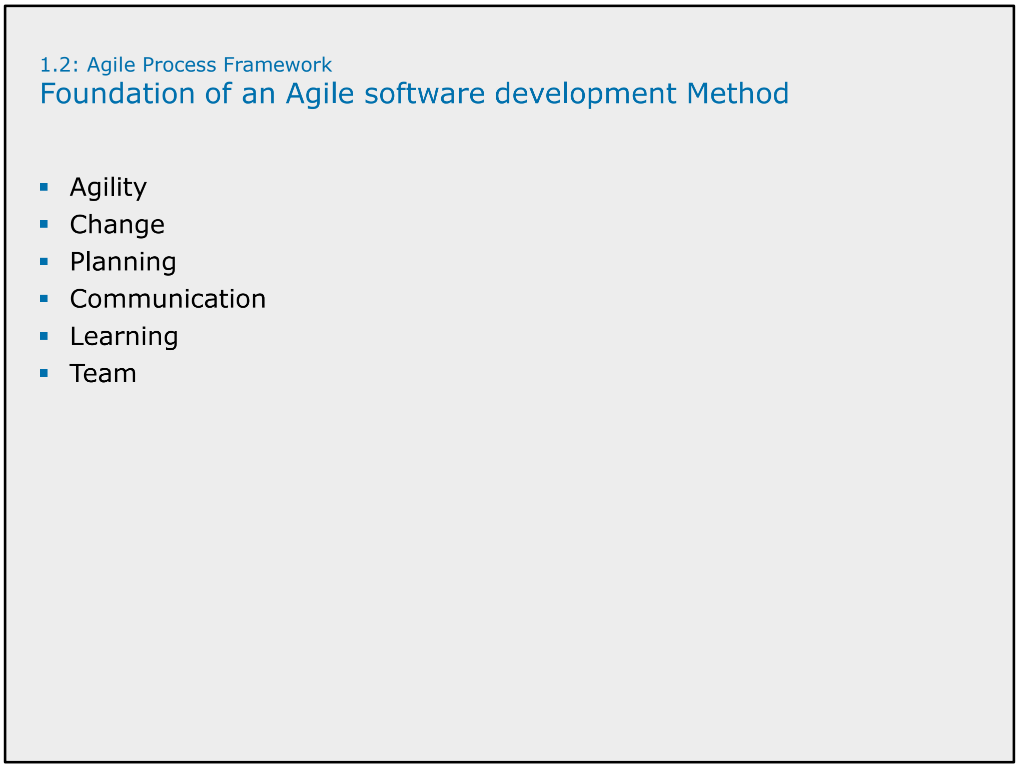
1.2: Agile Process Framework

What is NOT an Agile software development?

- Compressing the project schedule
- Eliminating all existing software development models
- Eliminating all documentation
- Writing code up to the last minute
- An excuse for doing nothing

What is NOT an Agile software development?

While there are many different definitions of what Agile Software Development is, there are reasons why each of the quoted sources view Agile in different ways. The term Agile is used to describe a software development approach that embodies the statements made in Agile Manifesto and follows the “12 Principles of Agile Methods”.
Many software development group says that they are using agile methods, but calling the development approach Agile doesn’t actually make it AGILE.
If you have implemented any of the above as part of your agile development approach, then you might need to rethink what agility actually is.



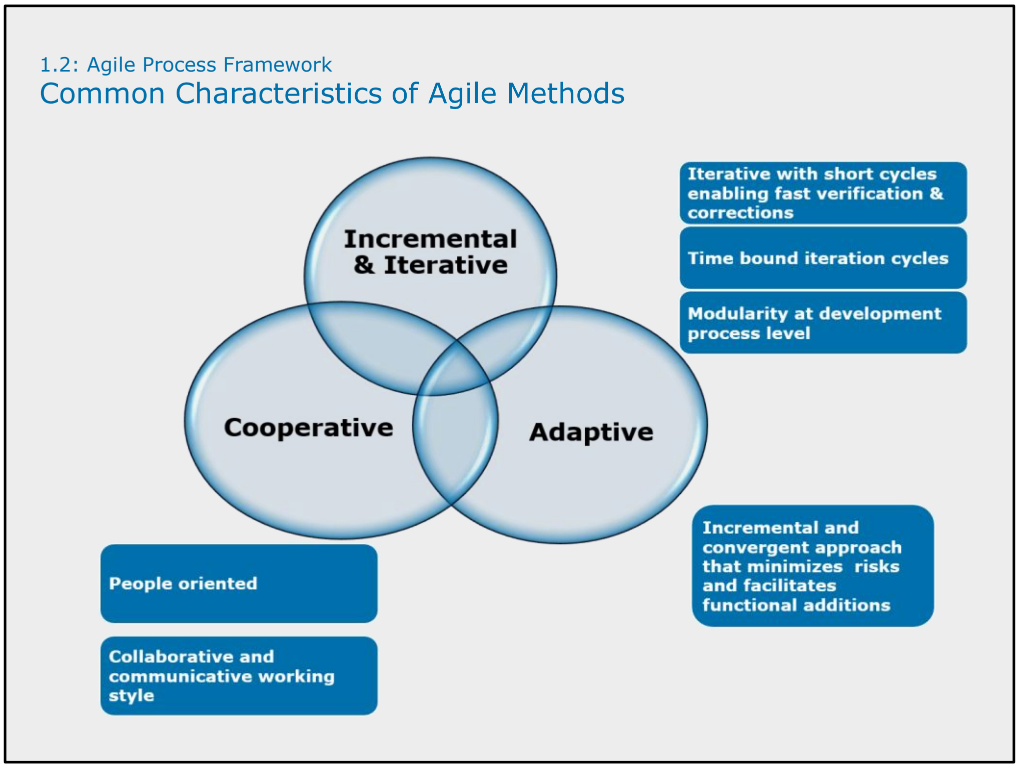
1.2: Agile Process Framework

Foundation of an Agile software development Method

- Agility
- Change
- Planning
- Communication
- Learning
- Team

Foundation of an Agile software development Method

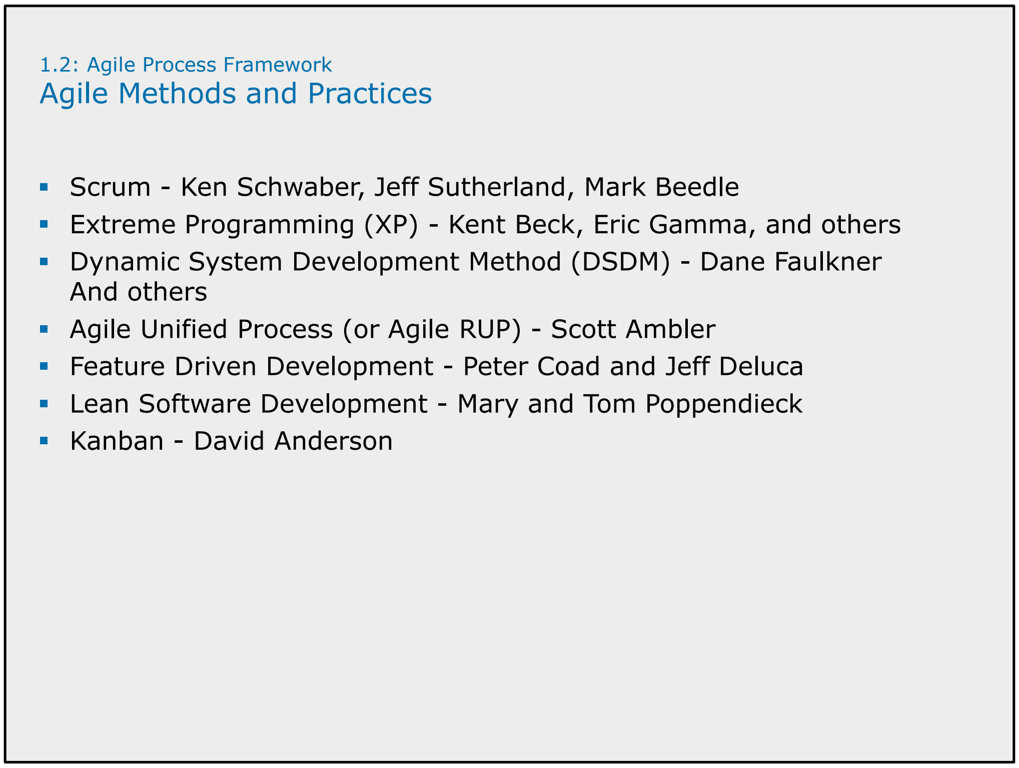
1. Agility – The ability to respond quickly.
2. Change – Is inevitable and important to be taken care of.
3. Planning – Creating small increments of the software system which is delivered as a series of time boxed iterations.
4. Communication – With the help of constant discussions & workshops aimed at information exchange between customer & team. This attribute is demonstrated via the daily stand-up meetings, collective iteration and release planning and the workshops held to flash out the user stories selected for each iteration.
5. Learning – Throughout the project cycle by analyzing and looking for the most appropriate ways to implement improvements based on the results of the previous iterations or release increments i.e. don’t repeat the same behavior if it didn’t result in the desired outcome.
6. Team – Authorized to take decisions and accountable for the release of the items committed to during the iteration planning.



1.2: Agile Process Framework

Common Characteristics of Agile Methods

Incremental & Iterative
Cooperative
Adaptive
Iterative with short cycles enabling fast verification & corrections
Time bound iteration cycles
Modularity at development process level
Incremental and convergent approach that minimizes risks and facilitates functional additions
People oriented
Collaborative and communicative working style



1.2: Agile Process Framework

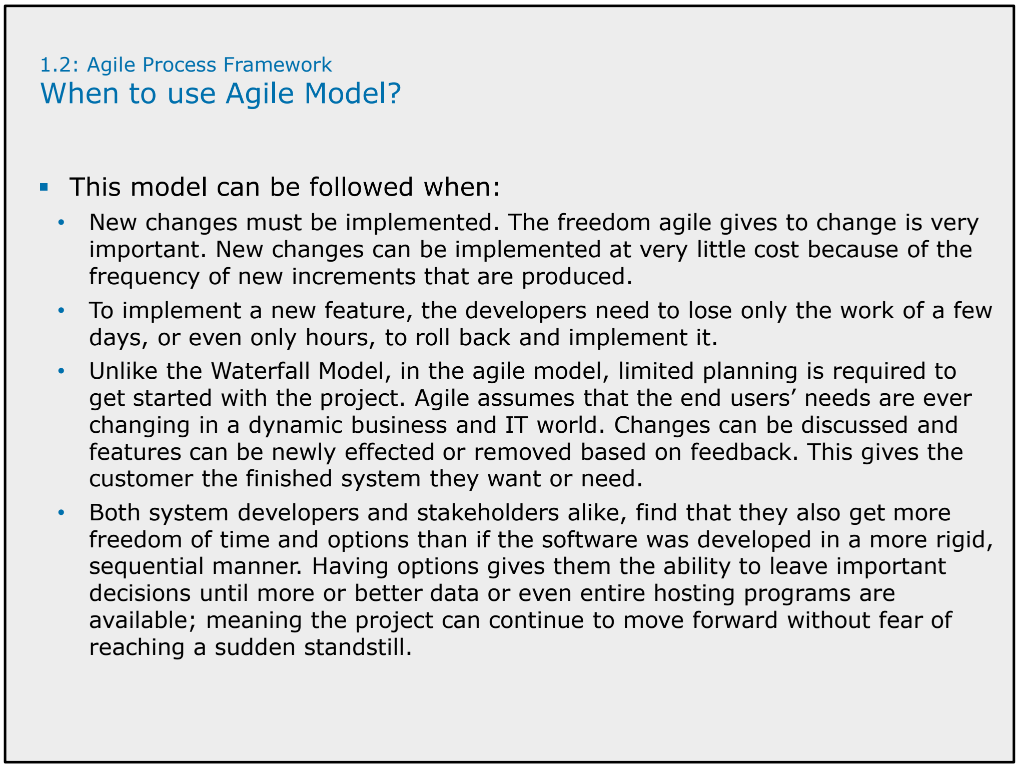
Agile Methods and Practices

- Scrum - Ken Schwaber, Jeff Sutherland, Mark Beedle
- Extreme Programming (XP) - Kent Beck, Eric Gamma, and others
- Dynamic System Development Method (DSDM) - Dane Faulkner And others
- Agile Unified Process (or Agile RUP) - Scott Ambler
- Feature Driven Development - Peter Coad and Jeff Deluca
- Lean Software Development - Mary and Tom Poppendieck
- Kanban - David Anderson

The term 'agile' is a philosophy and is a conceptual framework for undertaking software engineering projects. Most agile methods attempt to minimize risk by developing software in short time boxes, called iterations. Each iteration is like a miniature software project of its own, and includes all of the tasks necessary to release the mini increment of new functionality planning, requirements analysis, design, coding, testing and documentation. While an iteration may not add enough functionality to warrant releasing the product, an agile software project intends to be capable of releasing new software at the end of every iteration. Under this broad umbrella sits many more specific approaches.

Some of the well-known agile software development methods are given below.

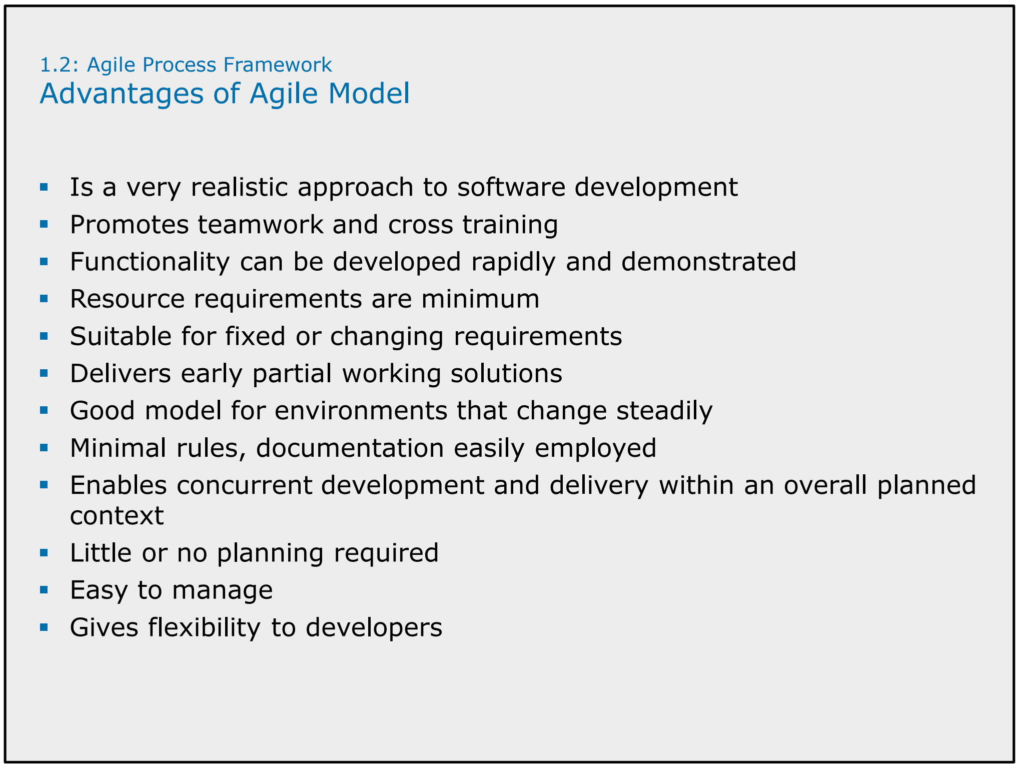
1. Scrum - Ken Schwaber, Jeff Sutherland, Mark Beedle
2. Extreme Programming (XP) - Kent Beck, Eric Gamma, and others
3. Dynamic System Development Method (DSDM) - Dane Faulkner And others
4. Agile Unified Process (or Agile RUP) - Scott Ambler
5. Feature Driven Development - Peter Coad and Jeff Deluca
6. Lean Software Development - Mary and Tom Poppendieck
7. Kanban - David Anderson



1.2: Agile Process Framework

When to use Agile Model?

- This model can be followed when:
  - New changes must be implemented. The freedom agile gives to change is very important. New changes can be implemented at very little cost because of the frequency of new increments that are produced.
  - To implement a new feature, the developers need to lose only the work of a few days, or even only hours, to roll back and implement it.
  - Unlike the Waterfall Model, in the agile model, limited planning is required to get started with the project. Agile assumes that the end users’ needs are ever changing in a dynamic business and IT world. Changes can be discussed and features can be newly effected or removed based on feedback. This gives the customer the finished system they want or need.
  - Both system developers and stakeholders alike, find that they also get more freedom of time and options than if the software was developed in a more rigid, sequential manner. Having options gives them the ability to leave important decisions until more or better data or even entire hosting programs are available; meaning the project can continue to move forward without fear of reaching a sudden standstill.



1.2: Agile Process Framework

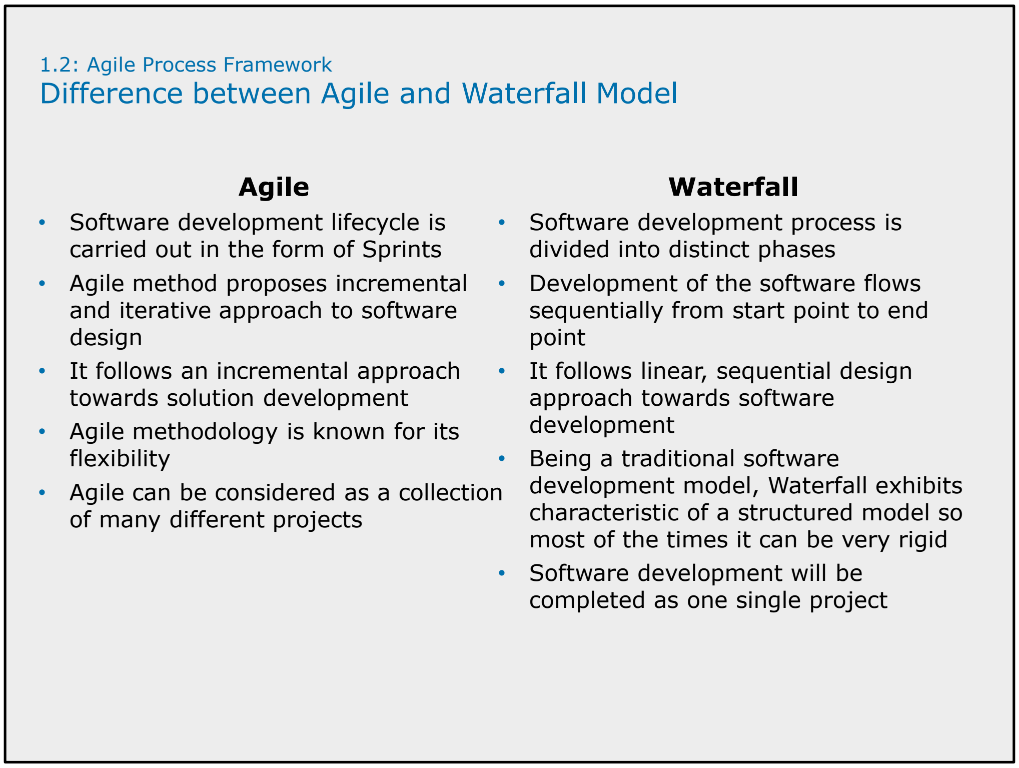
Advantages of Agile Model

- Is a very realistic approach to software development
- Promotes teamwork and cross training
- Functionality can be developed rapidly and demonstrated
- Resource requirements are minimum
- Suitable for fixed or changing requirements
- Delivers early partial working solutions
- Good model for environments that change steadily
- Minimal rules, documentation easily employed
- Enables concurrent development and delivery within an overall planned context
- Little or no planning required
- Easy to manage
- Gives flexibility to developers

1.2: Agile Process Framework

Disadvantages of Agile Model

- Not suitable for handling complex dependencies
- More risk of sustainability, maintainability and extensibility
- An overall plan, an agile leader and agile PM practice is a must without which it will not work
- Strict delivery management dictates the scope, functionality to be delivered, and adjustments to meet the deadlines
- Depends heavily on customer interaction, so if customer is not clear, team can be driven in the wrong direction
- There is very high individual dependency, since there is minimum documentation generated
- Transfer of technology to new team members may be quite challenging due to lack of documentation



1.2: Agile Process Framework

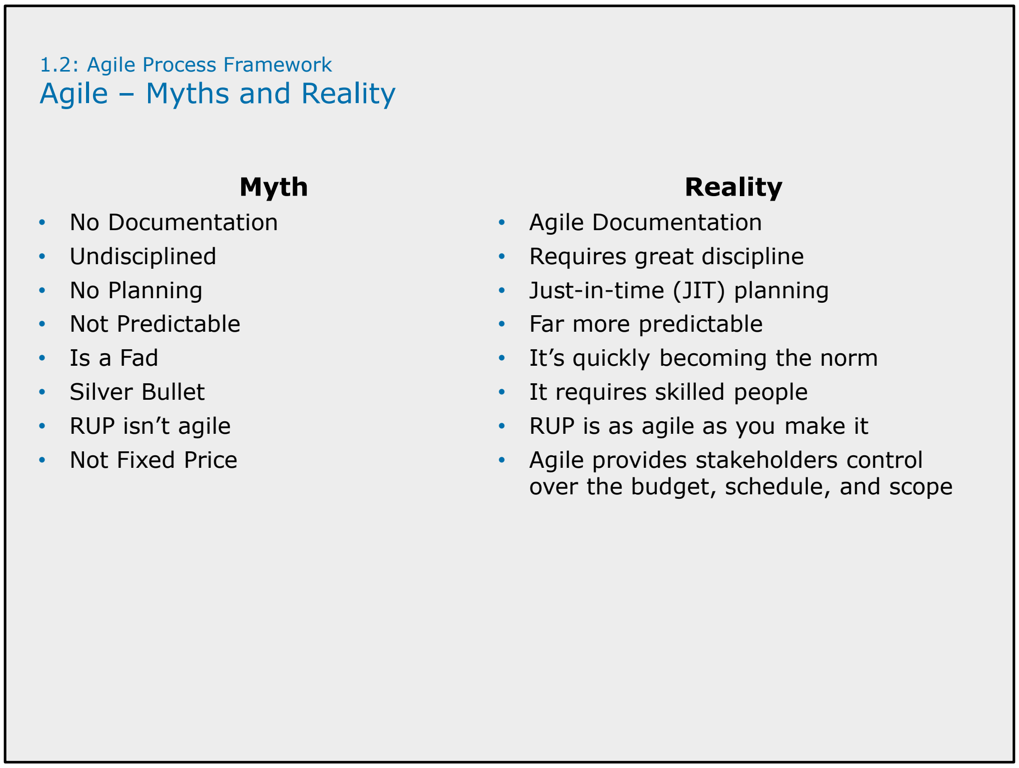
Difference between Agile and Waterfall Model

Agile

- Software development lifecycle is carried out in the form of Sprints
- Agile method proposes incremental and iterative approach to software design
- It follows an incremental approach towards solution development
- Agile methodology is known for its flexibility
- Agile can be considered as a collection of many different projects

Waterfall

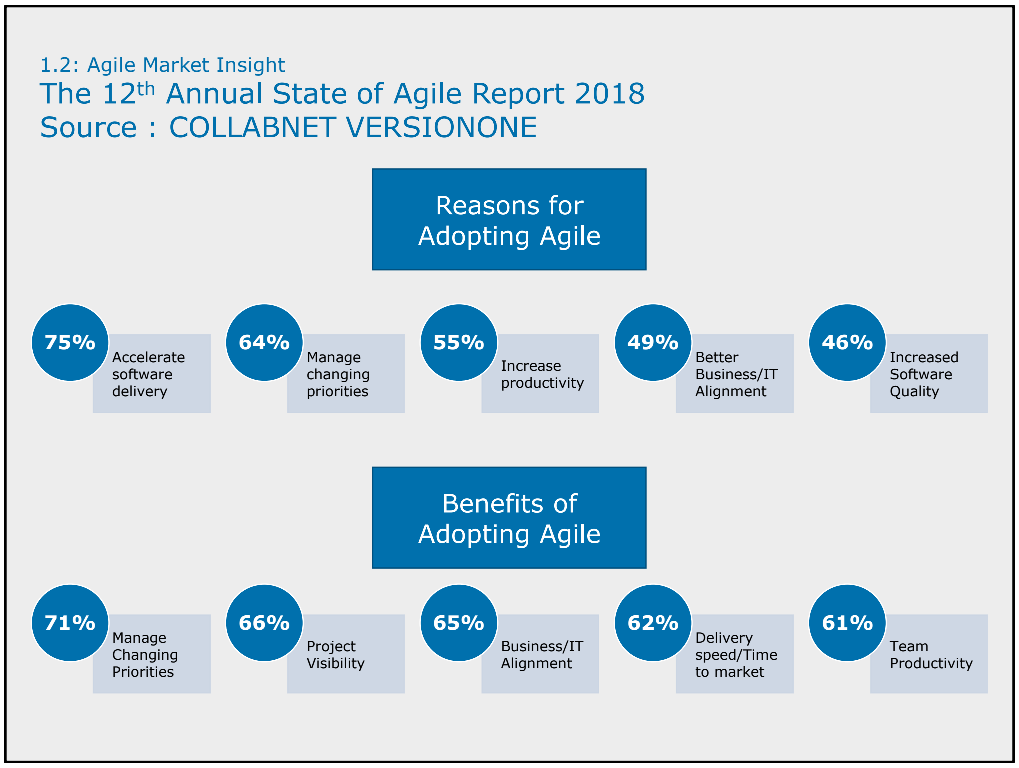
- Software development process is divided into distinct phases
- Development of the software flows sequentially from start point to end point
- It follows linear, sequential design approach towards software development
- Being a traditional software development model, Waterfall exhibits characteristic of a structured model so most of the times it can be very rigid
- Software development will be completed as one single project



1.2: Agile Process Framework

Agile – Myths and Reality

Myth	Reality
No Documentation	Agile Documentation
Undisciplined	Requires great discipline
No Planning	Just-in-time (JIT) planning
Not Predictable	Far more predictable
Is a Fad	It’s quickly becoming the norm
Silver Bullet	It requires skilled people
RUP isn’t agile	RUP is as agile as you make it
Not Fixed Price	Agile provides stakeholders control over the budget, schedule, and scope



1.2: Agile Market Insight

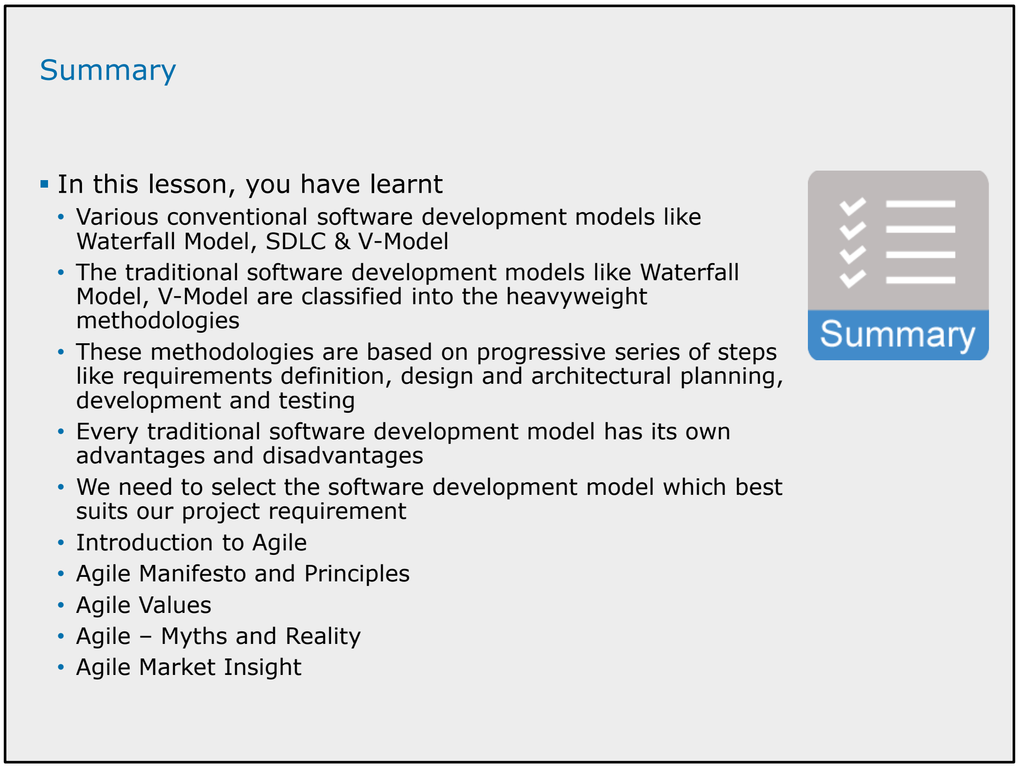
The 12th Annual State of Agile Report 2018 Source : COLLABNET VERSIONONE

Reasons for Adopting Agile

- **75%** Accelerate software delivery
- **64%** Manage changing priorities
- **55%** Increase productivity
- **49%** Better Business/IT Alignment
- **46%** Increased Software Quality

Benefits of Adopting Agile

- **71%** Manage Changing Priorities
- **66%** Project Visibility
- **65%** Business/IT Alignment
- **62%** Delivery speed/Time to market
- **61%** Team Productivity



Summary

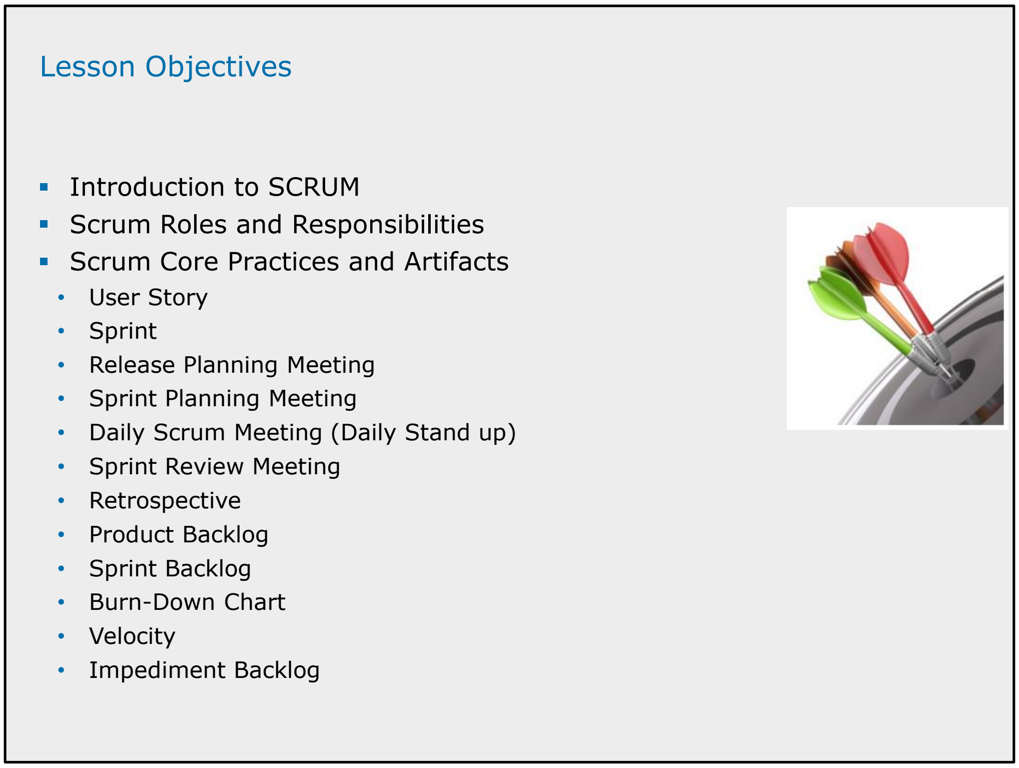
- In this lesson, you have learnt
  - Various conventional software development models like Waterfall Model, SDLC & V-Model
  - The traditional software development models like Waterfall Model, V-Model are classified into the heavyweight methodologies
  - These methodologies are based on progressive series of steps like requirements definition, design and architectural planning, development and testing
  - Every traditional software development model has its own advantages and disadvantages
  - We need to select the software development model which best suits our project requirement
  - Introduction to Agile
  - Agile Manifesto and Principles
  - Agile Values
  - Agile – Myths and Reality
  - Agile Market Insight

Summary

Add the notes here.

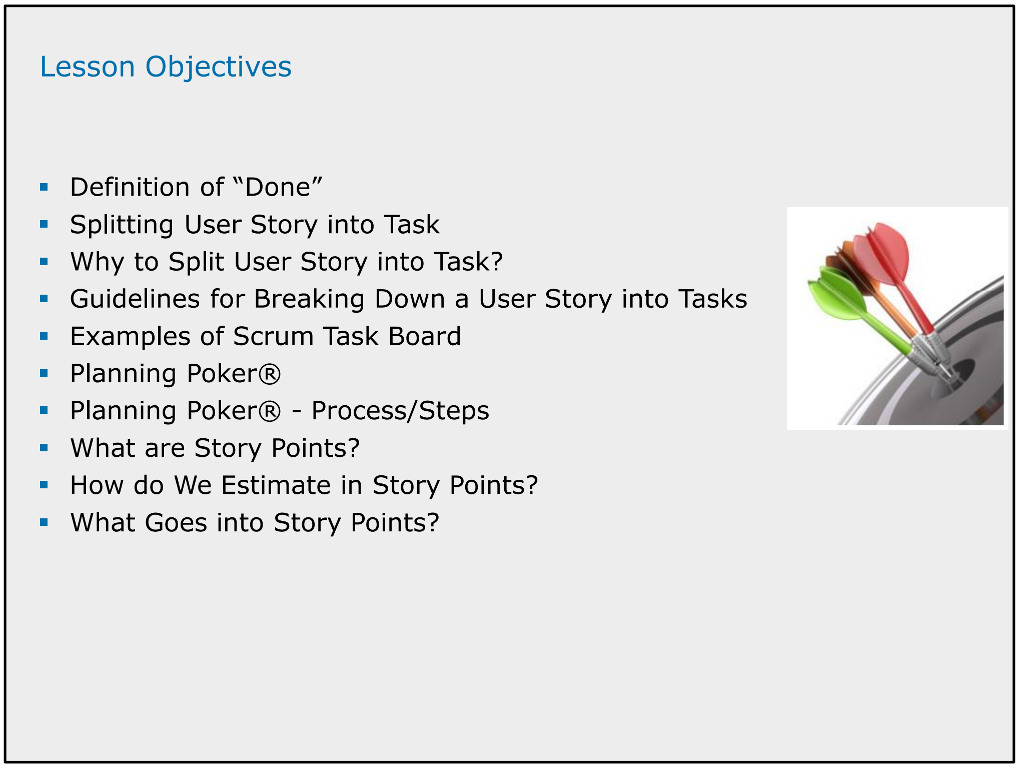
Introduction to Agile

Lesson 2: Agile Methods and Practices - SCRUM



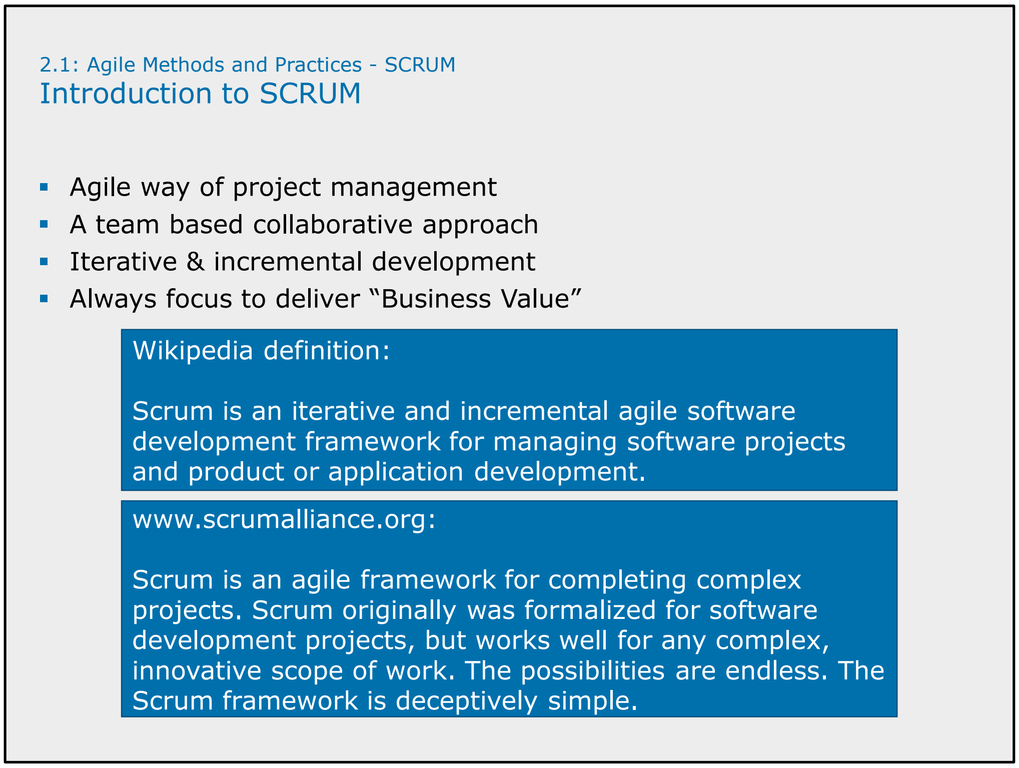
Lesson Objectives

- Introduction to SCRUM
- Scrum Roles and Responsibilities
- Scrum Core Practices and Artifacts
  - User Story
  - Sprint
  - Release Planning Meeting
  - Sprint Planning Meeting
  - Daily Scrum Meeting (Daily Stand up)
  - Sprint Review Meeting
  - Retrospective
  - Product Backlog
  - Sprint Backlog
  - Burn-Down Chart
  - Velocity
  - Impediment Backlog



Lesson Objectives

- Definition of “Done”
- Splitting User Story into Task
- Why to Split User Story into Task?
- Guidelines for Breaking Down a User Story into Tasks
- Examples of Scrum Task Board
- Planning Poker®
- Planning Poker® - Process/Steps
- What are Story Points?
- How do We Estimate in Story Points?
- What Goes into Story Points?



2.1: Agile Methods and Practices - SCRUM

Introduction to SCRUM

- Agile way of project management
- A team based collaborative approach
- Iterative & incremental development
- Always focus to deliver "Business Value"

Wikipedia definition:

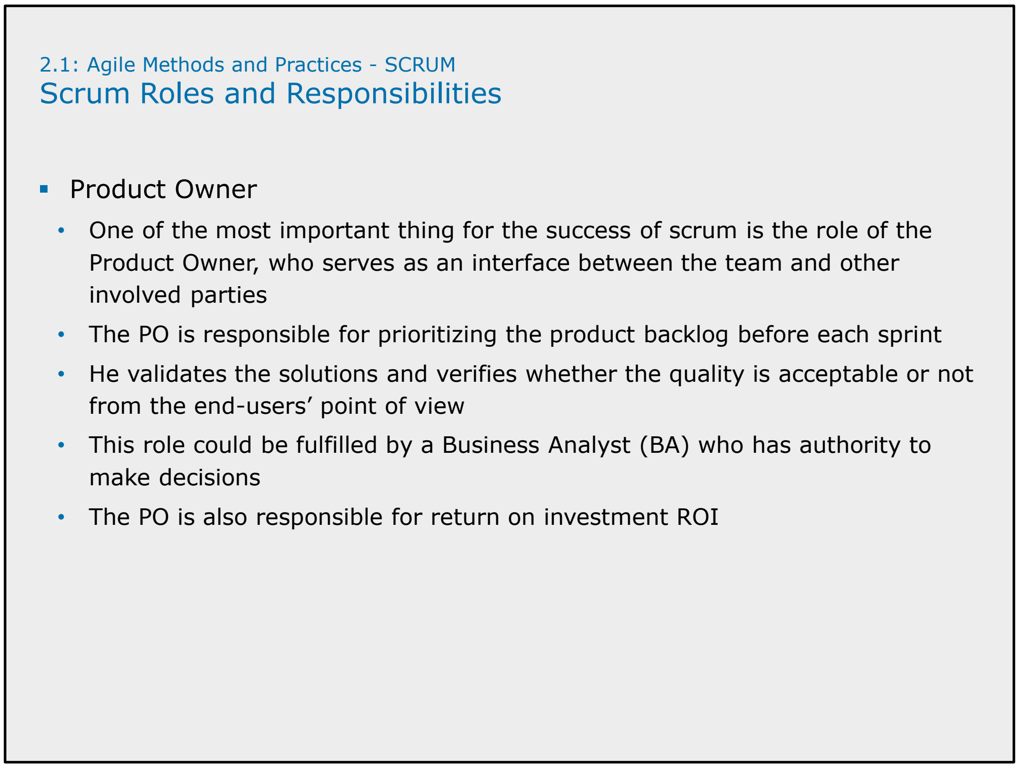
Scrum is an iterative and incremental agile software development framework for managing software projects and product or application development.

www.scrumalliance.org:

Scrum is an agile framework for completing complex projects. Scrum originally was formalized for software development projects, but works well for any complex, innovative scope of work. The possibilities are endless. The Scrum framework is deceptively simple.

Introduction to SCRUM

Scrum is one of the more popular agile methods in use today and originally developed at Easel in 1993 by Jeff Sutherland and Ken Schwaber. Since that time it has been used at many software companies, which have resulted in the method being extended and enhanced.
When Jeff Sutherland created the scrum process in 1993, he borrowed the term "scrum" from an analogy put forth in a 1986 study by Takeuchi and Nonaka, published in the Harvard Business Review. In that study, Takeuchi and Nonaka compare high-performing, cross-functional teams to the scrum formation used by Rugby teams.
Scrum is the leading agile development methodology, used by Fortune 500 companies around the world. The Scrum Alliance exists to transform the way we tackle complex projects, bringing the Scrum framework and agile principles beyond software development to the broader world of work.



2.1: Agile Methods and Practices - SCRUM

Scrum Roles and Responsibilities

- Product Owner
  - One of the most important thing for the success of scrum is the role of the Product Owner, who serves as an interface between the team and other involved parties
  - The PO is responsible for prioritizing the product backlog before each sprint
  - He validates the solutions and verifies whether the quality is acceptable or not from the end-users' point of view
  - This role could be fulfilled by a Business Analyst (BA) who has authority to make decisions
  - The PO is also responsible for return on investment ROI

Scrum Roles and Responsibilities:

In contrast to classical project management methods, Scrum doesn't have and doesn't need a product manager, a task manager or a team leader. The most important three roles of Scrum are:

Product Owner
Scrum Master
Development team

These three roles are coequal and all of them have certain responsibilities.

2.1: Agile Methods and Practices - SCRUM

Scrum Roles and Responsibilities (Cont.)

- Scrum Master
  - Responsible for implementing scrum practices and rules and ensuring that it is followed
  - They are also responsible for removing "roadblocks‟
  - They are not responsible for managing and controlling the team
  - The Scrum Master does not interfere into the decisions of the team regarding specifically the development, but rather is there for the team as an advisor
  - The Scrum Master has to create an optimal working-condition for the team and is responsible for this condition to be retained, in order to meet the goals of every sprint

2.1: Agile Methods and Practices - SCRUM

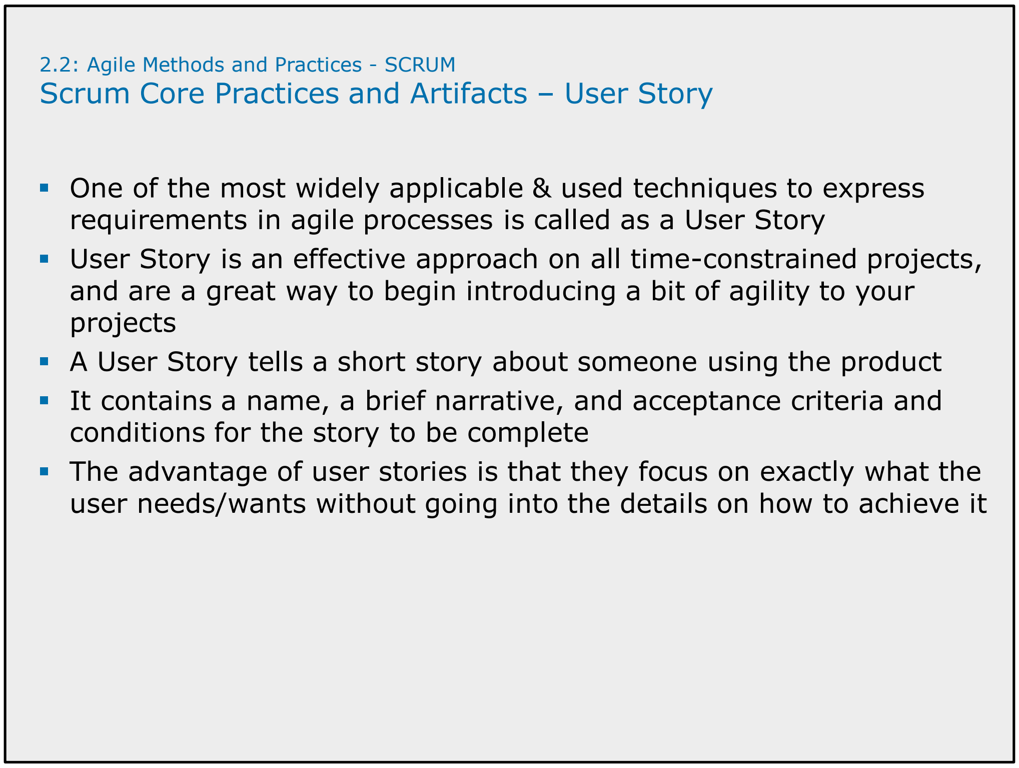
Scrum Roles and Responsibilities (Cont.)

- The Team
  - They are all responsible for implementing the functionality through the delivery of the product
  - The "team" consists of developers, testers, Quality Assurance, BA and any other people who are required to implement and deliver functionality
  - Different from other methods, in Scrum a team is not only the set of people those who receives their tasks from the project leader, it rather decides for itself which requirement or User Stories it can accomplish in one sprint

2.2: Agile Methods and Practices - SCRUM

Scrum Core Practices and Artifacts

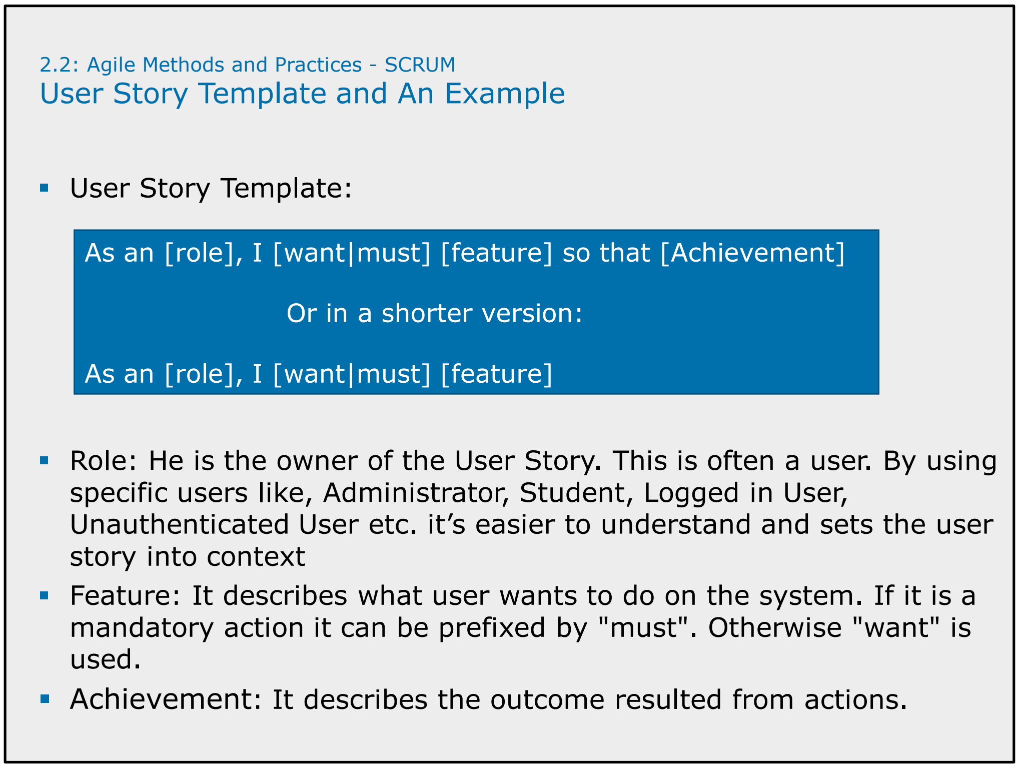
- User Story
- Sprint
- Release Planning Meeting
- Sprint Planning Meeting
- Daily Scrum Meeting (Daily Stand up)
- Sprint Review Meeting
- Retrospective
- Product Backlog
- Sprint Backlog
- Burn-Down Chart
- Velocity
- Impediment Backlog



2.2: Agile Methods and Practices - SCRUM

Scrum Core Practices and Artifacts – User Story

- One of the most widely applicable & used techniques to express requirements in agile processes is called as a User Story
- User Story is an effective approach on all time-constrained projects, and are a great way to begin introducing a bit of agility to your projects
- A User Story tells a short story about someone using the product
- It contains a name, a brief narrative, and acceptance criteria and conditions for the story to be complete
- The advantage of user stories is that they focus on exactly what the user needs/wants without going into the details on how to achieve it



2.2: Agile Methods and Practices - SCRUM

User Story Template and An Example

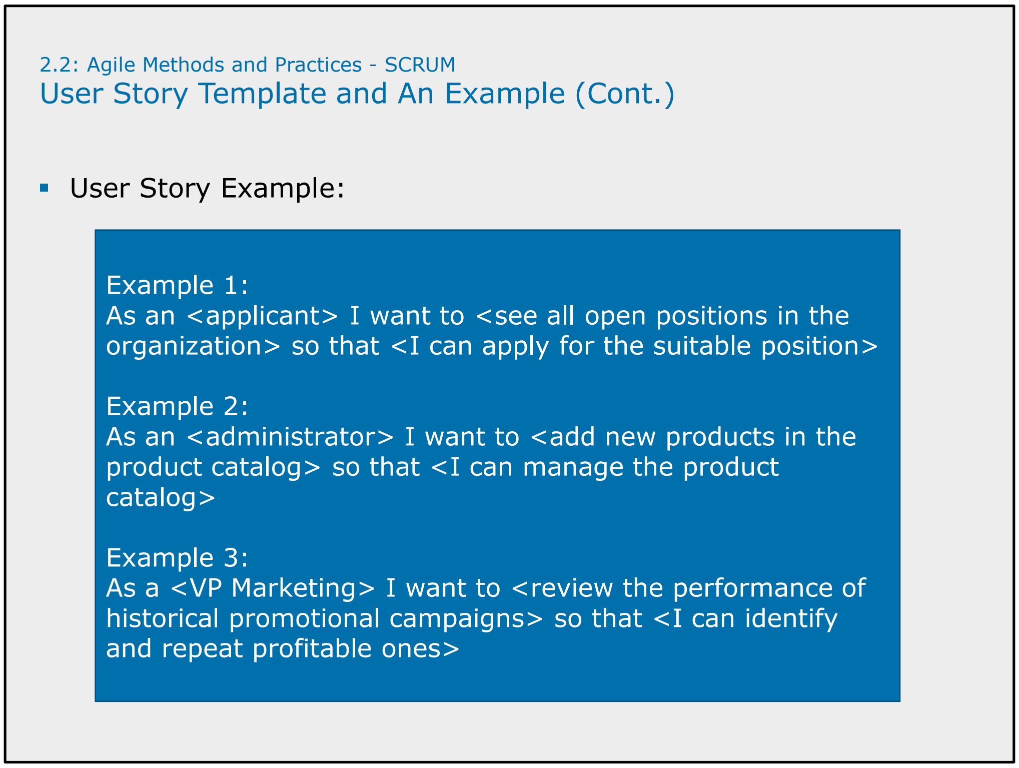
- User Story Template:

  As an [role], I [want|must] [feature] so that [Achievement]

  Or in a shorter version:

  As an [role], I [want|must] [feature]

- Role: He is the owner of the User Story. This is often a user. By using specific users like, Administrator, Student, Logged in User, Unauthenticated User etc. it’s easier to understand and sets the user story into context
- Feature: It describes what user wants to do on the system. If it is a mandatory action it can be prefixed by "must". Otherwise "want" is used.
- Achievement: It describes the outcome resulted from actions.



2.2: Agile Methods and Practices - SCRUM

User Story Template and An Example (Cont.)

- User Story Example:

Example 1:
As an <applicant> I want to <see all open positions in the organization> so that <I can apply for the suitable position>

Example 2:
As an <administrator> I want to <add new products in the product catalog> so that <I can manage the product catalog>

Example 3:
As a <VP Marketing> I want to <review the performance of historical promotional campaigns> so that <I can identify and repeat profitable ones>

Class Exercise:

- Writing User Stories

2.2: Agile Methods and Practices - SCRUM

Scrum Core Practices and Artifacts – Sprint

- In the Scrum method of agile software development, work is confined to a regular, repeatable work cycle, known as a sprint or iteration
- In by-the-book Scrum, a sprint is 30 days long, but many teams prefer shorter sprints, such as one-week, two-week, or three-week sprints
- During each sprint, a team creates a shippable product, no matter how basic that product is
- Each sprint is a small and manageable iteration
- It contains design, development, testing, and documentation
- A new Sprint starts immediately after the conclusion of the previous Sprint
- Sprints contain and consist of the Sprint Planning, Daily Scrums, the development work, the Sprint Review, and the Sprint Retrospective
- The sprint begins with Planning and ends with Review & Retrospective

Introduction to Sprint

Each Sprint may be considered a project with no more than a one-month duration. Like projects, Sprints are used to accomplish something. Each Sprint has a goal of what is to be built, a design and flexible plan that will guide building it, the work, and the resultant product increment.
A Sprint would be cancelled if the Sprint Goal becomes obsolete. This might occur if the company changes direction or if market or technology conditions change. In general, a Sprint should be cancelled if it no longer makes sense given the circumstances. But, due to the short duration of Sprints, cancellation rarely makes sense.

2.2: Agile Methods and Practices - SCRUM

Scrum Core Practices and Artifacts – Sprint (Cont.)

- Additionally a Sprint Retrospective Meeting is conducted to check and improve the project execution processes: What was good during the Sprint, what should continue as it is and what should be improved
- During the Sprint a short daily Standup-Meeting (Daily Scrum Meeting) is held to update the status of the items and to help self-organization of the team
- A Sprint can be cancelled before the Sprint time-box is over
- Only the Product Owner has the authority to cancel the Sprint, although he or she may do so under influence from the stakeholders, the Development Team, or the Scrum Master

2.2: Agile Methods and Practices - SCRUM

Scrum Core Practices and Artifacts – Release Planning

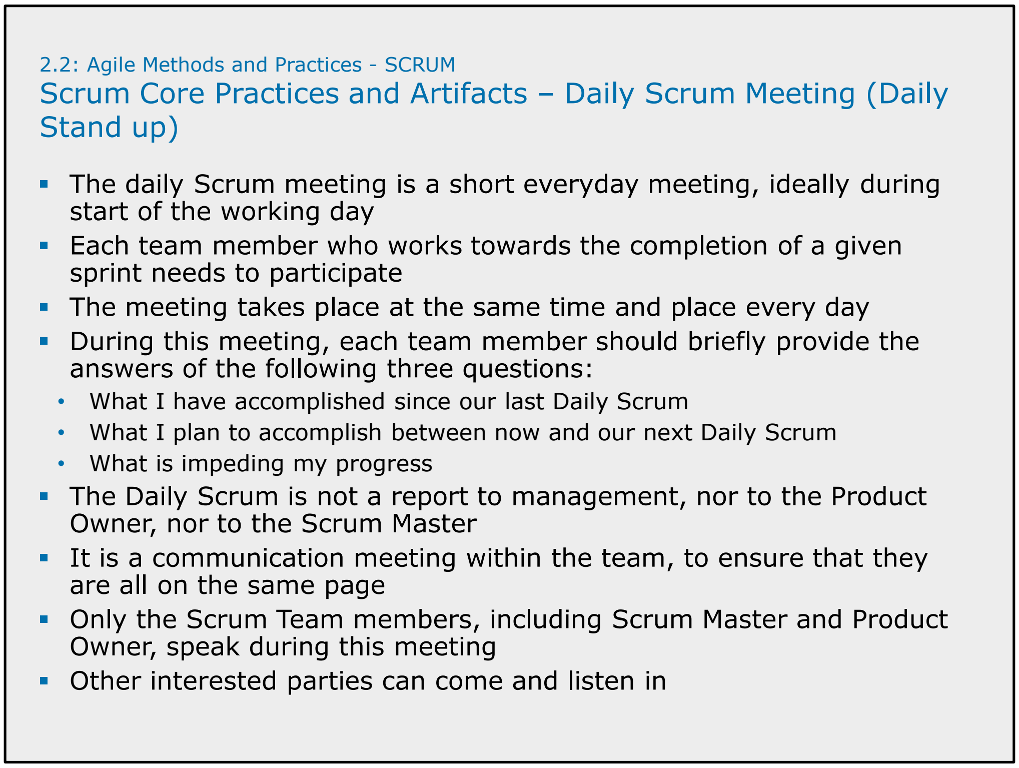
- The Product Owner, Scrum Master and Scrum Team defines the next release and the functionality at the release planning meeting
- The meeting may be divided between understanding the requirements and the acceptance criteria
- The exercise of splitting the users stories into product features is also carried out in the release planning meeting
- It is a guideline that reflects expectations about which features will be implemented and when they are completed

- Following are the prerequisites for Release Planning
  - A prioritized and estimated Scrum Product Backlog
  - The velocity (How much work the team can get done per iteration) of the Scrum Team
  - Goals for the schedule, scope, resource

2.2: Agile Methods and Practices - SCRUM

Scrum Core Practices and Artifacts – Sprint Planning Meeting

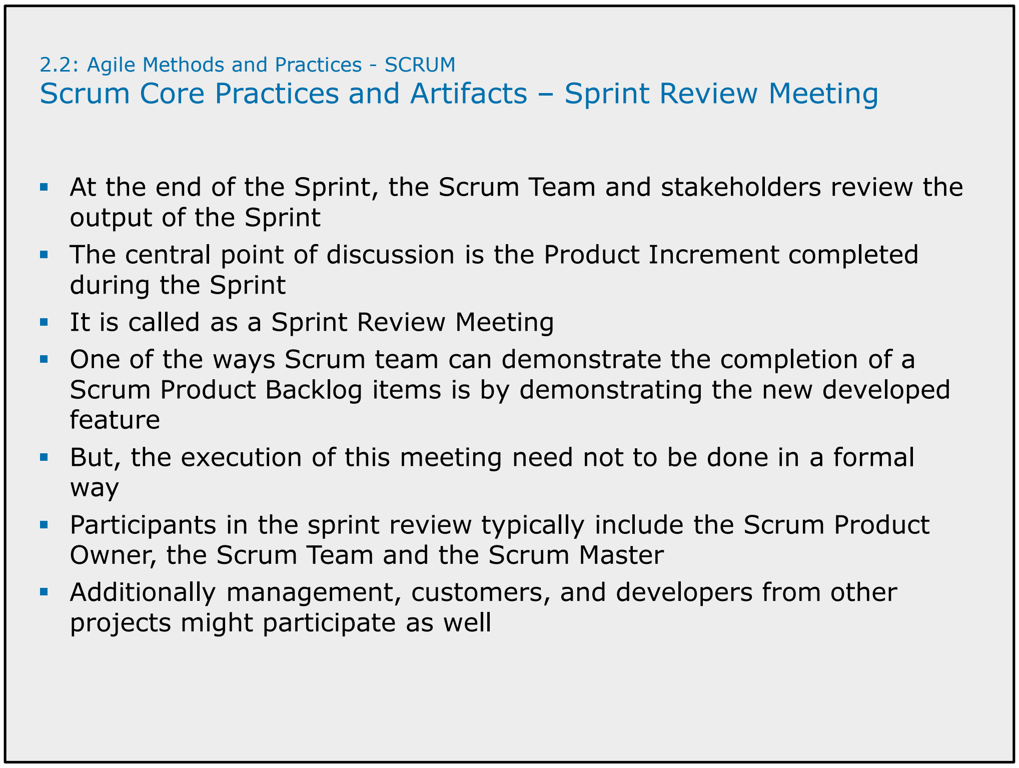
- Each Sprint begins with a time-boxed meeting called Sprint Planning
- Product Owner, Scrum Master and the Scrum team determine the next sprint goal and functionality at the Sprint Planning Meeting
- The team then devises the individual tasks that must be performed to build the product increment
- All Scrum meetings are time boxed
- The recommended time for the Sprint Planning meeting is two hours or less per week of Sprint duration
- Goal of this meeting is to define a realistic Sprint Backlog containing all items that could be fully implemented until the end of the Sprint



2.2: Agile Methods and Practices - SCRUM

Scrum Core Practices and Artifacts – Daily Scrum Meeting (Daily Stand up)

- The daily Scrum meeting is a short everyday meeting, ideally during start of the working day
- Each team member who works towards the completion of a given sprint needs to participate
- The meeting takes place at the same time and place every day
- During this meeting, each team member should briefly provide the answers of the following three questions:
  - What I have accomplished since our last Daily Scrum
  - What I plan to accomplish between now and our next Daily Scrum
  - What is impeding my progress
- The Daily Scrum is not a report to management, nor to the Product Owner, nor to the Scrum Master
- It is a communication meeting within the team, to ensure that they are all on the same page
- Only the Scrum Team members, including Scrum Master and Product Owner, speak during this meeting
- Other interested parties can come and listen in



2.2: Agile Methods and Practices - SCRUM

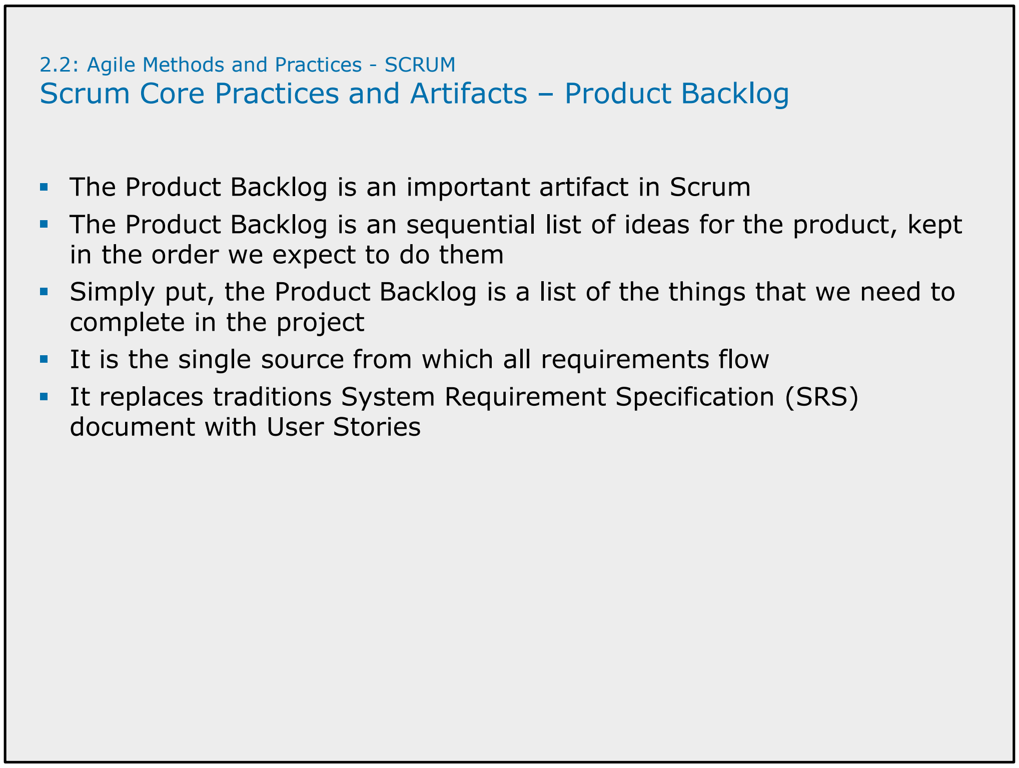
Scrum Core Practices and Artifacts – Sprint Review Meeting

- At the end of the Sprint, the Scrum Team and stakeholders review the output of the Sprint
- The central point of discussion is the Product Increment completed during the Sprint
- It is called as a Sprint Review Meeting
- One of the ways Scrum team can demonstrate the completion of a Scrum Product Backlog items is by demonstrating the new developed feature
- But, the execution of this meeting need not to be done in a formal way
- Participants in the sprint review typically include the Scrum Product Owner, the Scrum Team and the Scrum Master
- Additionally management, customers, and developers from other projects might participate as well

2.2: Agile Methods and Practices - SCRUM

Scrum Core Practices and Artifacts – Sprint Retrospective

- At the end of each Sprint, the Scrum team meets for the Sprint Retrospective
- The purpose is to review how things went with respect to the process, the relationships among people, and the tools
- The team identifies what went well and not so well, and identifies potential improvements
- They come up with a plan for improving things in the future
- All Scrum meetings are time-boxed
- The recommended time box for the Sprint Retrospective is one hour per week of Sprint duration
- Retrospective enables Scrum teams to focus on overall performance of the team and enhance the output of Sprint



2.2: Agile Methods and Practices - SCRUM

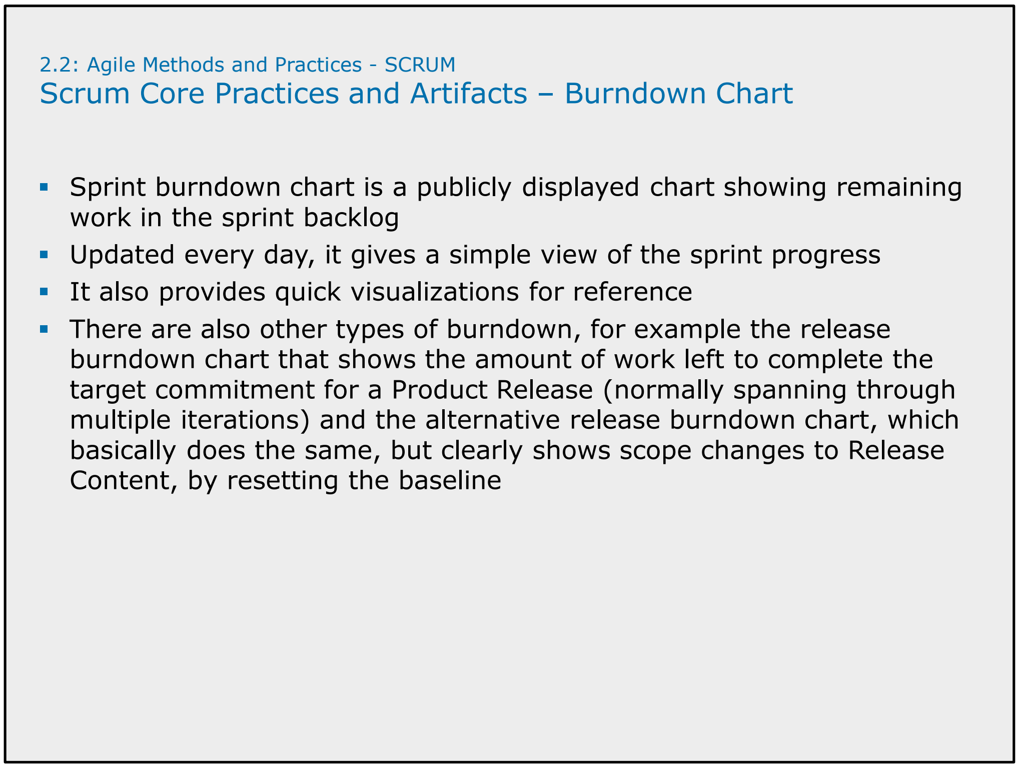
Scrum Core Practices and Artifacts – Product Backlog

- The Product Backlog is an important artifact in Scrum
- The Product Backlog is an sequential list of ideas for the product, kept in the order we expect to do them
- Simply put, the Product Backlog is a list of the things that we need to complete in the project
- It is the single source from which all requirements flow
- It replaces traditions System Requirement Specification (SRS) document with User Stories

2.2: Agile Methods and Practices - SCRUM

Scrum Core Practices and Artifacts – Sprint Backlog

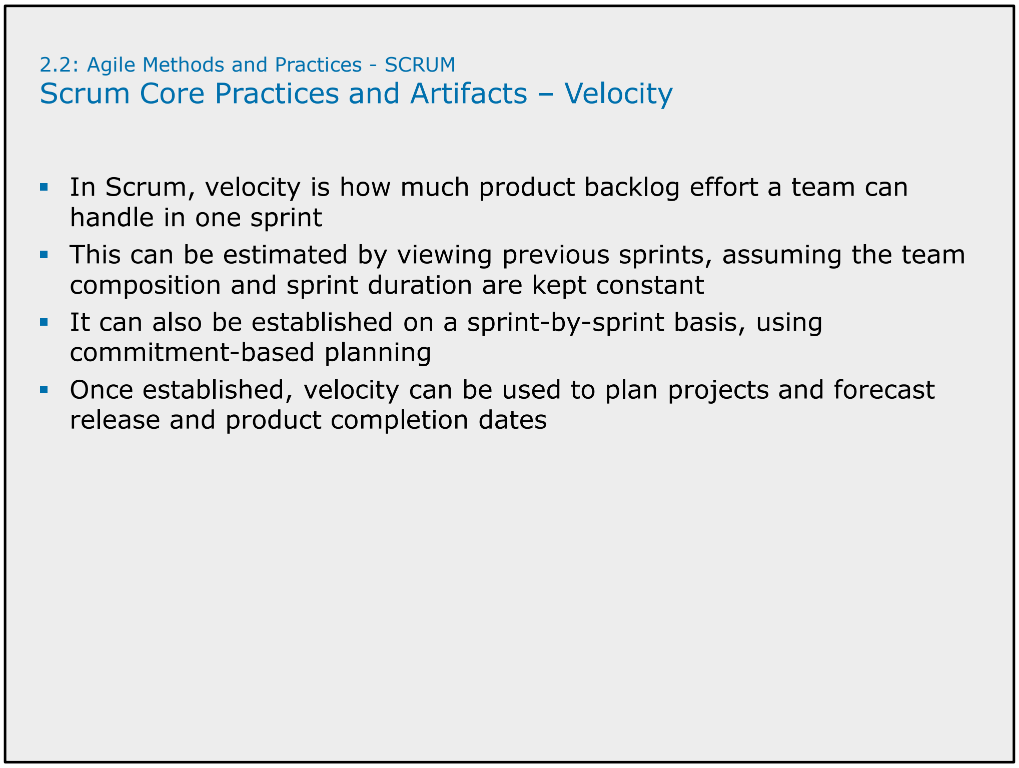
- Sprint Backlog is the list of refined Product Backlog items chosen for development in the current Sprint, together with the team's plan for accomplishing the work
- It reflects the team's forecast of what work can be completed
- With the Sprint Backlog in place, the Sprint begins, and the Development Team develops the new Product Increment defined by the Sprint Backlog
- Sprint Backlog can be kept electronically within e.g. an Excel-Sheet or with cards on a task board



2.2: Agile Methods and Practices - SCRUM

Scrum Core Practices and Artifacts – Burndown Chart

- Sprint burndown chart is a publicly displayed chart showing remaining work in the sprint backlog
- Updated every day, it gives a simple view of the sprint progress
- It also provides quick visualizations for reference
- There are also other types of burndown, for example the release burndown chart that shows the amount of work left to complete the target commitment for a Product Release (normally spanning through multiple iterations) and the alternative release burndown chart, which basically does the same, but clearly shows scope changes to Release Content, by resetting the baseline



2.2: Agile Methods and Practices - SCRUM

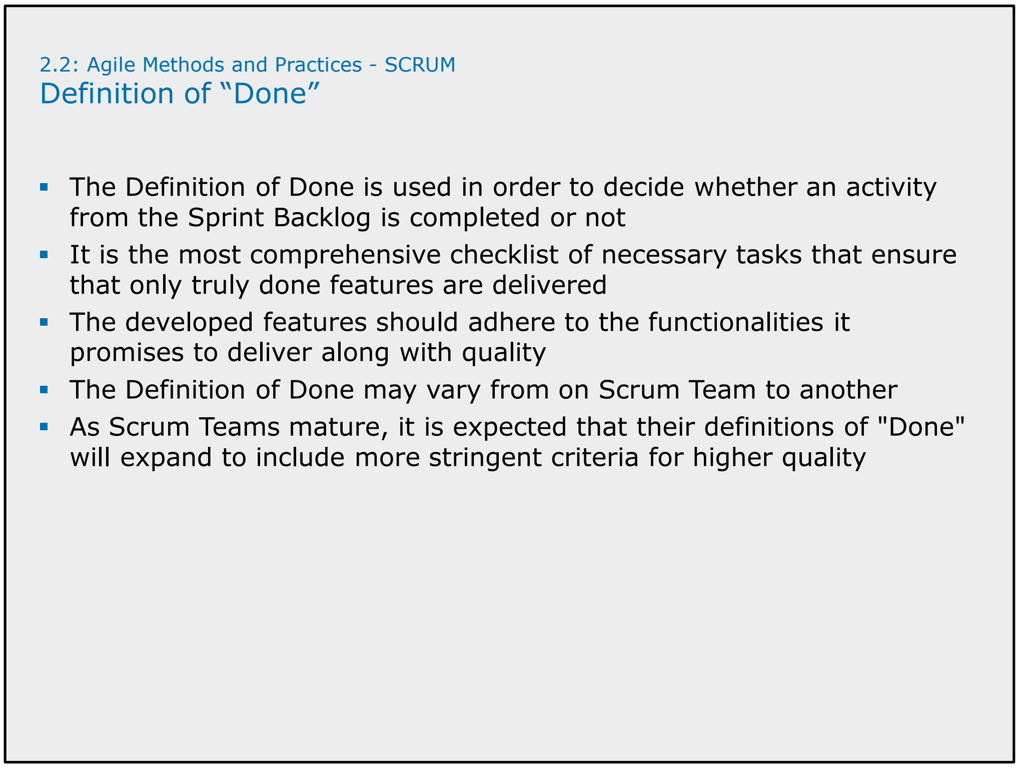
Scrum Core Practices and Artifacts – Velocity

- In Scrum, velocity is how much product backlog effort a team can handle in one sprint
- This can be estimated by viewing previous sprints, assuming the team composition and sprint duration are kept constant
- It can also be established on a sprint-by-sprint basis, using commitment-based planning
- Once established, velocity can be used to plan projects and forecast release and product completion dates

2.2: Agile Methods and Practices - SCRUM

Scrum Core Practices and Artifacts – Impediment Backlog

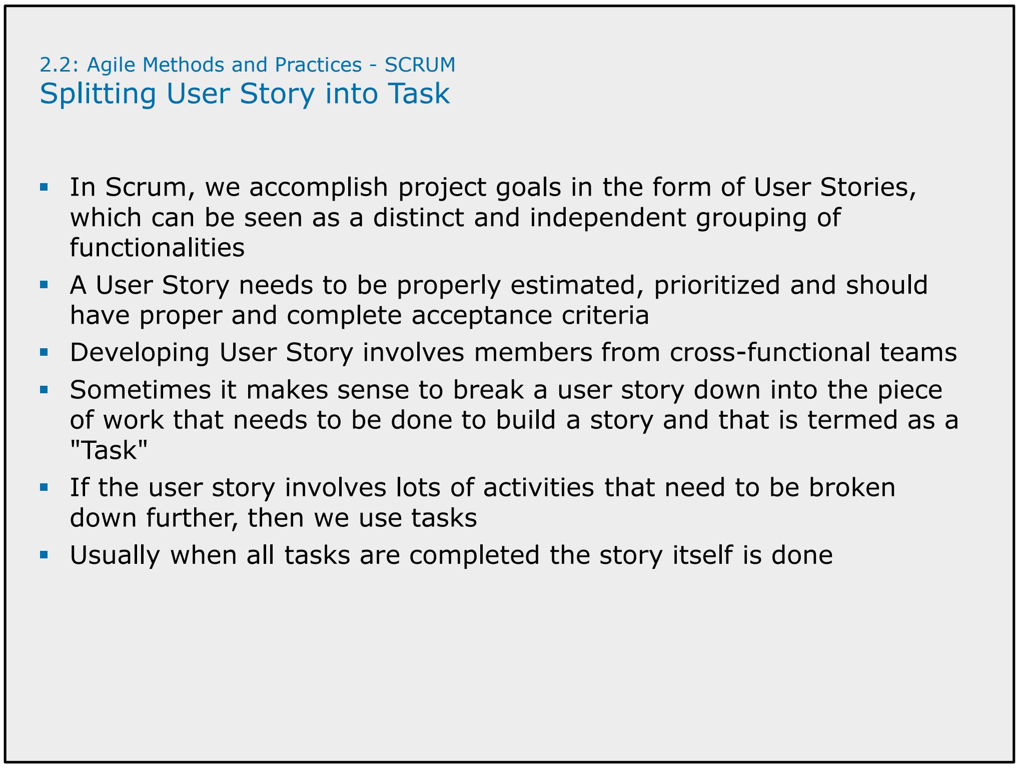
- In the process of developing a product or service, there always be the constant set of challenges a Scrum team needs to face and these challenges generally have the complete potential to impede the Scrum team's performance and delivery of the product or service being developed
- These challenges should be recorded and prioritized and the artifact used by the Scrum team for the same is called as Impediment Backlog
- Impediment Backlog is one of the artifacts, which is often overlooked
- A Scrum Master should be doing all that is required and within their power to ensure nothing is stopping the team from achieving their Sprint goals
- Scrum Master should be ensuring that impediments get resolved as fast as possible, ideally within the day that they are raised



2.2: Agile Methods and Practices - SCRUM

Definition of “Done”

- The Definition of Done is used in order to decide whether an activity from the Sprint Backlog is completed or not
- It is the most comprehensive checklist of necessary tasks that ensure that only truly done features are delivered
- The developed features should adhere to the functionalities it promises to deliver along with quality
- The Definition of Done may vary from on Scrum Team to another
- As Scrum Teams mature, it is expected that their definitions of "Done" will expand to include more stringent criteria for higher quality



2.2: Agile Methods and Practices - SCRUM

Splitting User Story into Task

- In Scrum, we accomplish project goals in the form of User Stories, which can be seen as a distinct and independent grouping of functionalities
- A User Story needs to be properly estimated, prioritized and should have proper and complete acceptance criteria
- Developing User Story involves members from cross-functional teams
- Sometimes it makes sense to break a user story down into the piece of work that needs to be done to build a story and that is termed as a "Task"
- If the user story involves lots of activities that need to be broken down further, then we use tasks
- Usually when all tasks are completed the story itself is done

2.2: Agile Methods and Practices - SCRUM

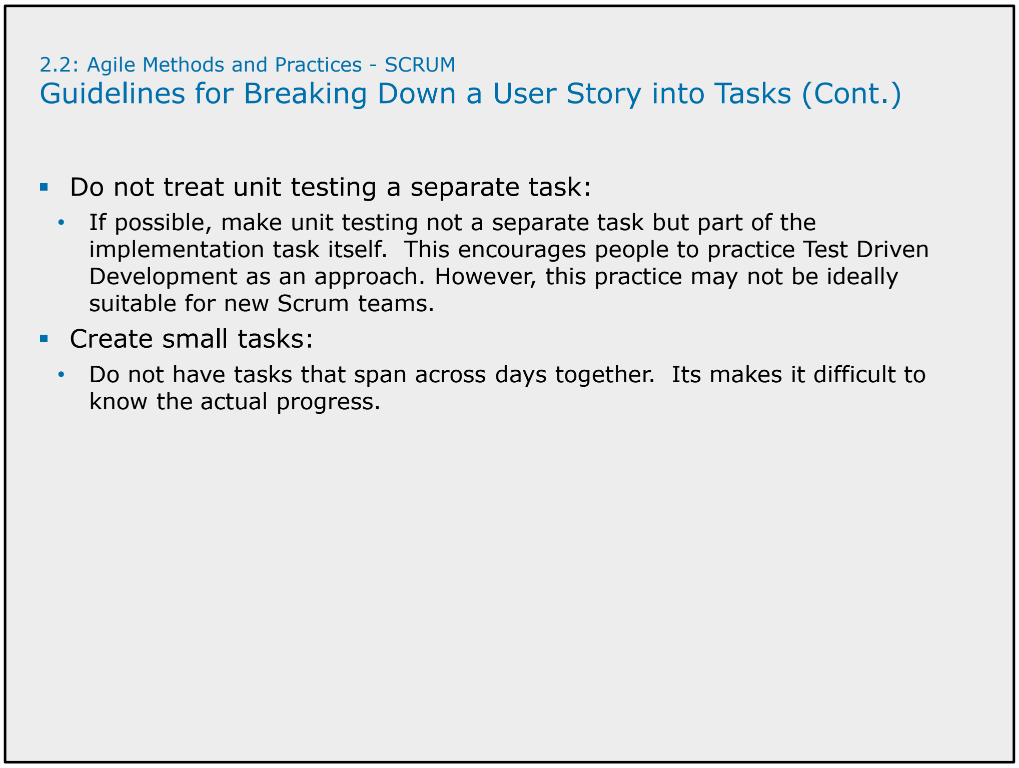
Why to Split User Story into Task?

- Splitting a User Story into tasks brings more understanding into team members about what is required to be done to accomplish the story
- Independent tasks will allow multiple developers to work on the same story in parallel, so it will be finished as fast as possible
- It allows teams to measure progress
- The tasks involved in each User Story are posted on task board
- Task board can be physical task board or electronic in case teams are distributed geographically
- developers will pick up tasks and move them from "not started" to "in progress" and finally to "done‟
- Even if all tasks do not represent exactly the same effort, we can reasonably think that when 3 tasks are done for a 6 task story, we are probably around half way

2.2: Agile Methods and Practices - SCRUM

Guidelines for Breaking Down a User Story into Tasks

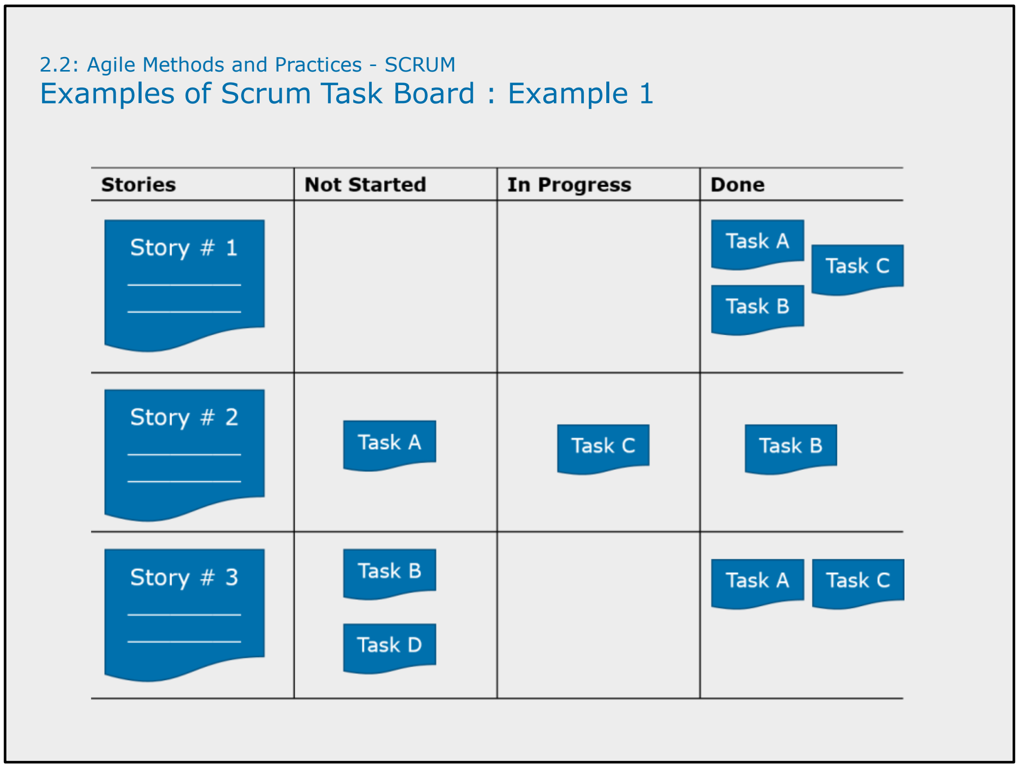
- Create meaningful tasks:
  - Elaborate tasks in such a way that they indicate & clearly convey the intent behind the task. For example, instead of saying Coding, describe the tasks as “Develop Registration Class”, “Develop the scripting part for registration functionality”, “Develop the automatic registration id generation for the registration functionality”, “Create the user table and save the registration data into the database” etc. Such tasks are more meaningful rather than just saying coding and testing.
- Task should be rightly sized:
  - Breaking the user stories with too many details is an overhead. For example, "Write code to accept UserName & Password", "Write code to validate login". Such tasks requires very small duration for completion but too many details at minute level can create unnecessary overheads. One guideline is to have tasks that span less than 8 hours so that each one of them can be completed in at least a day.



2.2: Agile Methods and Practices - SCRUM

Guidelines for Breaking Down a User Story into Tasks (Cont.)

- Do not treat unit testing a separate task:
  - If possible, make unit testing not a separate task but part of the implementation task itself. This encourages people to practice Test Driven Development as an approach. However, this practice may not be ideally suitable for new Scrum teams.
- Create small tasks:
  - Do not have tasks that span across days together. Its makes it difficult to know the actual progress.



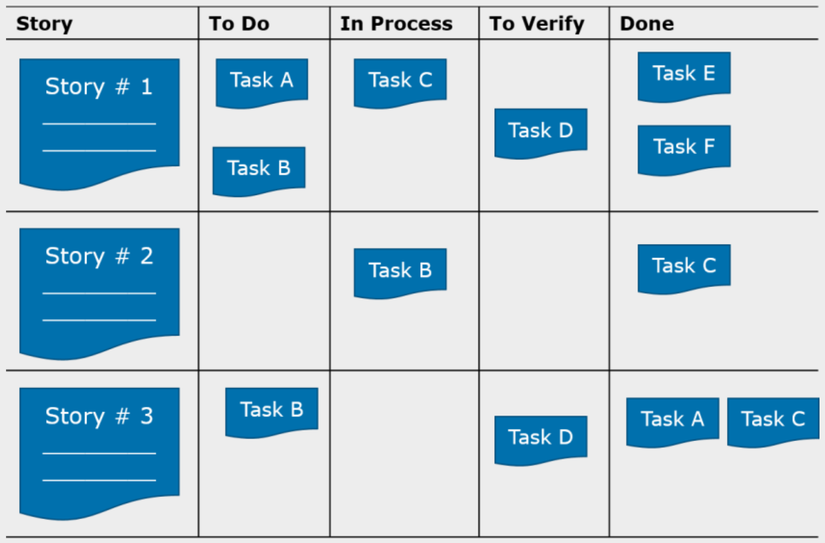
2.2: Agile Methods and Practices - SCRUM

Examples of Scrum Task Board : Example 1

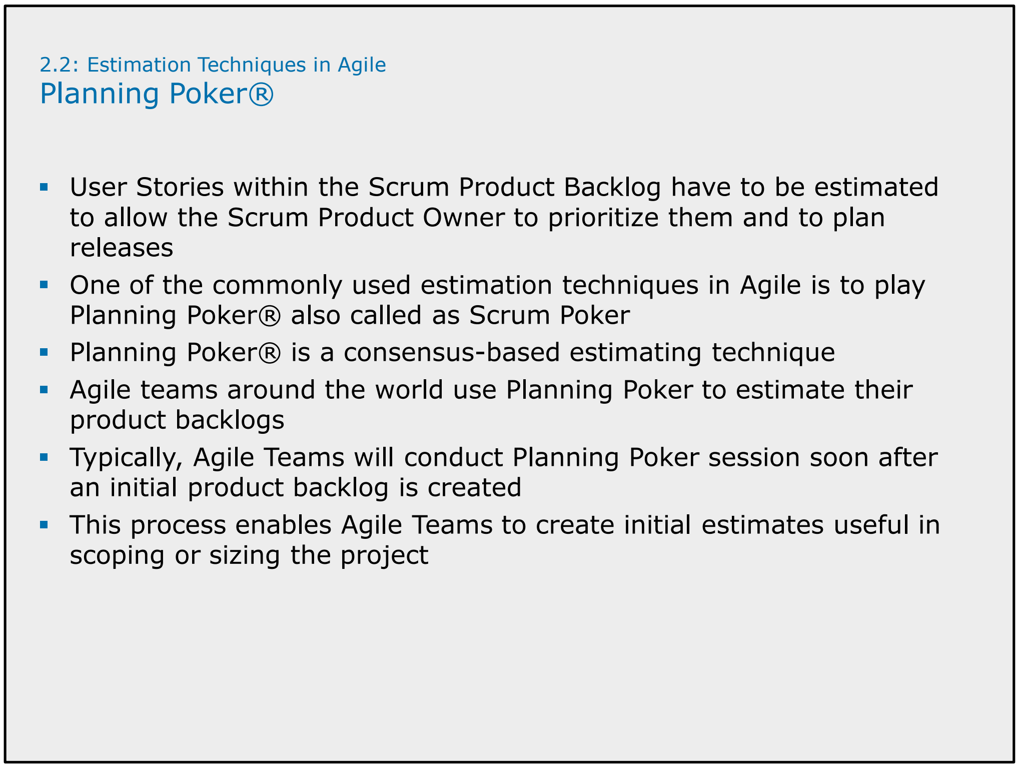
Stories	Not Started	In Progress	Done
Story # 1			Task A, Task C, Task B
Story # 2	Task A	Task C	Task B
Story # 3	Task B, Task D		Task A, Task C

2.2: Agile Methods and Practices - SCRUM

Examples of Scrum Task Board : Example 2



Story	To Do	In Process	To Verify	Done
Story # 1	Task A, Task B	Task C	Task D	Task E, Task F
Story # 2		Task B		Task C
Story # 3	Task B		Task D	Task A, Task C



2.2: Estimation Techniques in Agile

Planning Poker®

- User Stories within the Scrum Product Backlog have to be estimated to allow the Scrum Product Owner to prioritize them and to plan releases
- One of the commonly used estimation techniques in Agile is to play Planning Poker® also called as Scrum Poker
- Planning Poker® is a consensus-based estimating technique
- Agile teams around the world use Planning Poker to estimate their product backlogs
- Typically, Agile Teams will conduct Planning Poker session soon after an initial product backlog is created
- This process enables Agile Teams to create initial estimates useful in scoping or sizing the project

2.2: Estimation Techniques in Agile

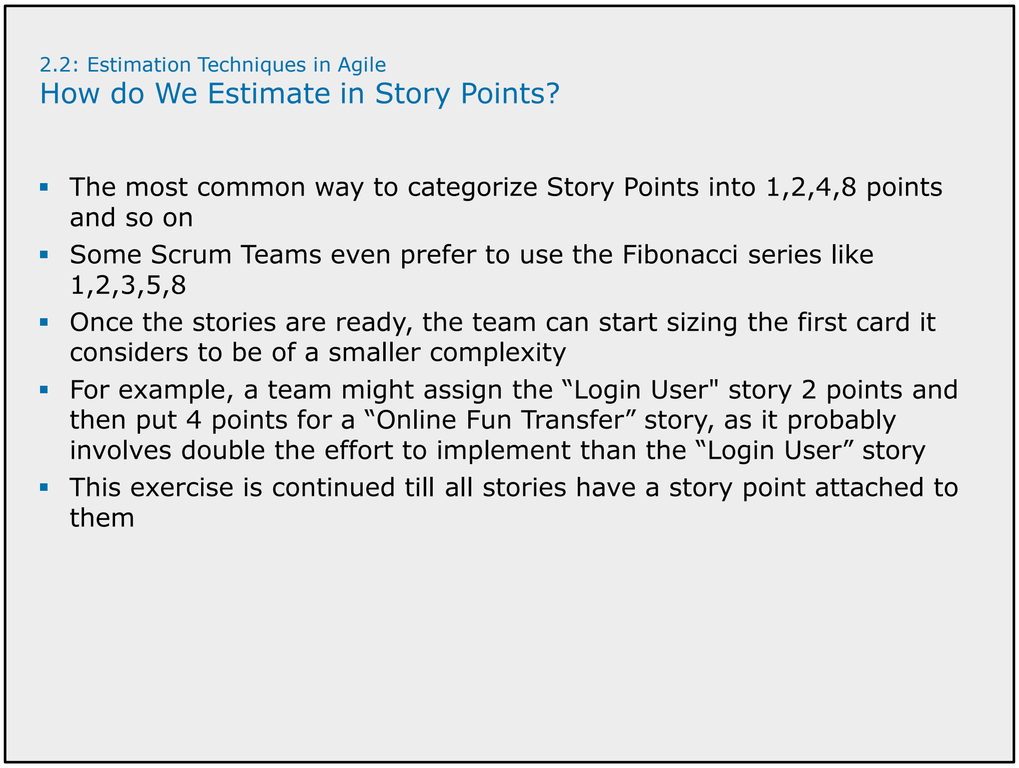
Planning Poker® - Process/Steps

- Following are the prerequisites to play Planning Poker®:
  - The list of User Stories to be estimated
  - Decks of numbered cards - Generally, the deck of cards used in estimation process contains cards showing the Fibonacci sequence including a zero: 0, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89 also other similar progressions can also be used
- Steps to play Planning Poker®
  - The Scrum Product Owner presents the story to be estimated
  - The Scrum Team asks questions and the Scrum Product Owner explains in more detail
  - If many stories have to be estimated a time-constraint might be set as well
  - If the time-constraint is reached and the Scrum Team does not understand the story it is a sign that the story has to be re-written
  - Each member of the Scrum Team privately chooses the card representing the estimation
  - After everyone has chosen a card, all selections are revealed
  - People with high and low estimates are allowed to explain their estimate
  - Estimation starts again until a consent is found
  - This game is repeated until all stories are estimated

2.2: Estimation Techniques in Agile

What are Story Points?

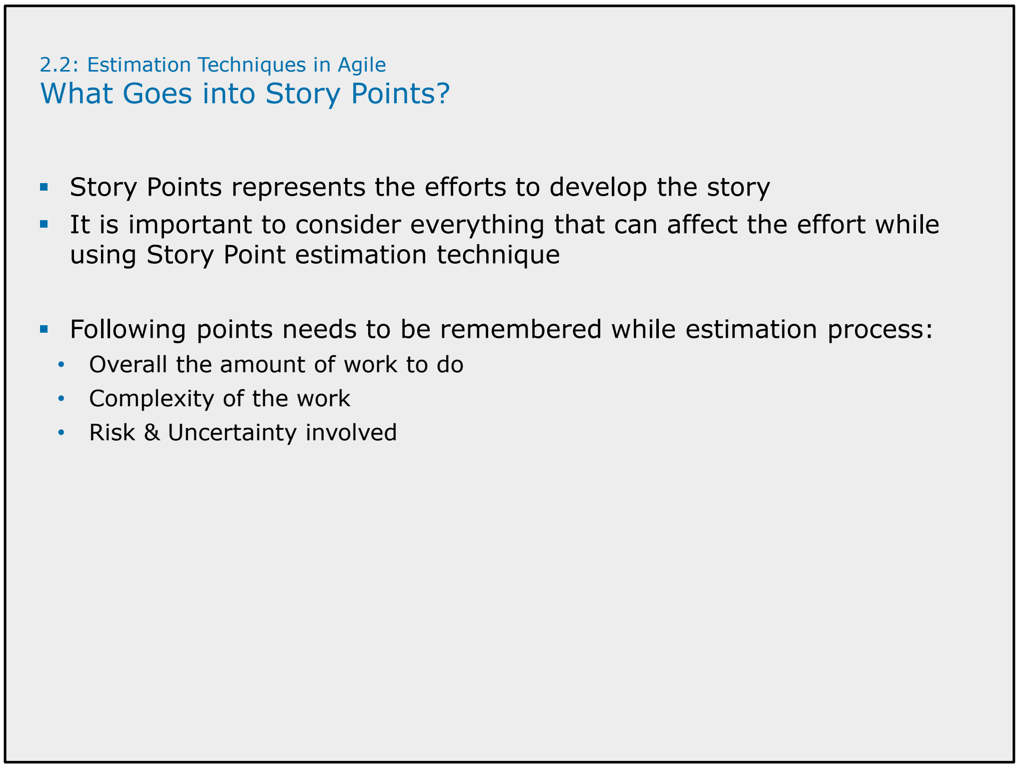
- Story points are a unit of measure for expressing an estimate of the overall effort that will be required to fully implement a product backlog item or any other piece of work
- When we estimate with story points, we assign a point value to each item
- Story point estimation is done using relative sizing by comparing one story with a sample set of previously sized stories
- Teams are able to estimate much more quickly without spending too much time in nailing down the exact number of hours or days required to finish a user story
- Ideally, the team who is responsible for realizing the story into solution should be part of the estimation process
- QA team should also be part of the estimation exercise if the story has additional testing efforts involved



2.2: Estimation Techniques in Agile

How do We Estimate in Story Points?

- The most common way to categorize Story Points into 1,2,4,8 points and so on
- Some Scrum Teams even prefer to use the Fibonacci series like 1,2,3,5,8
- Once the stories are ready, the team can start sizing the first card it considers to be of a smaller complexity
- For example, a team might assign the “Login User" story 2 points and then put 4 points for a “Online Fun Transfer” story, as it probably involves double the effort to implement than the “Login User” story
- This exercise is continued till all stories have a story point attached to them



2.2: Estimation Techniques in Agile

What Goes into Story Points?

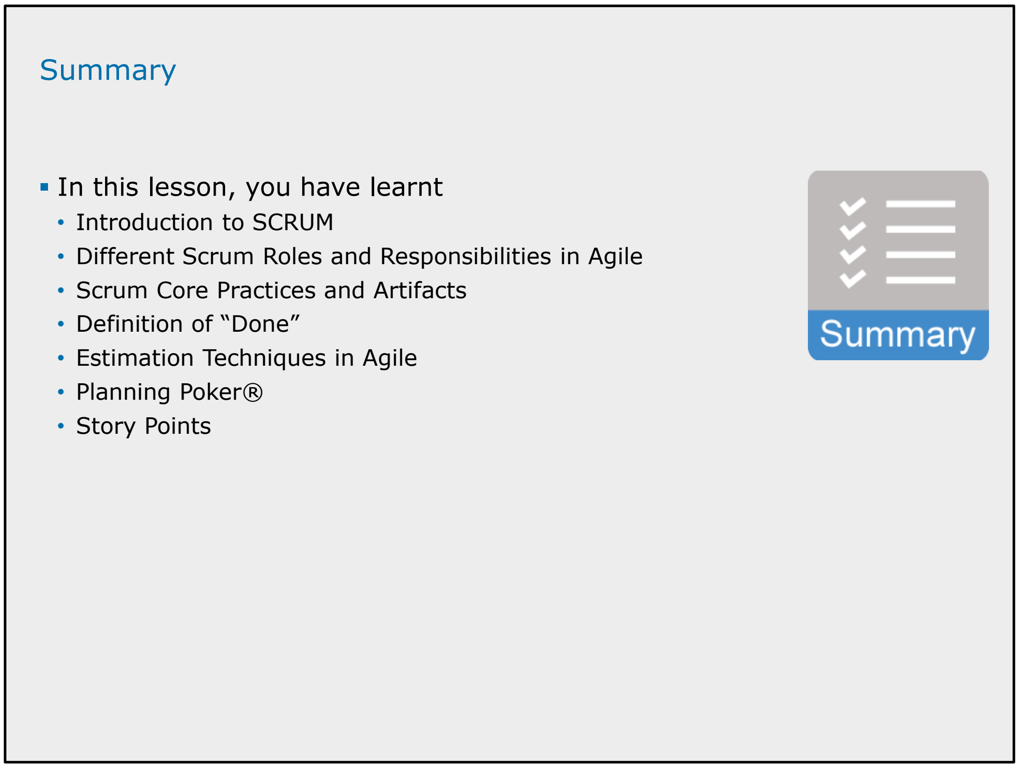
- Story Points represents the efforts to develop the story
- It is important to consider everything that can affect the effort while using Story Point estimation technique
- Following points needs to be remembered while estimation process:
  - Overall the amount of work to do
  - Complexity of the work
  - Risk & Uncertainty involved

Following points needs to be remembered while estimation process:

1. Overall the amount of work to do: Certainly, if there is more to do of something, the estimate of effort should be larger. Consider the case of developing two web pages. The first page has only one field and a label asking to enter a name. The second page has 100 fields to simply be filled with a text. The only difference between these two pages is that there is more to do on the second page. The second page should be given more story points. It probably doesn't get 100 times more points even though there are 100 times as many fields. There are, after all, economies of scale and maybe making the second page is only 2 or 3 or 10 times as much effort as the first page.

2. Complexity of the work: Complexity should also be considered when providing a story point estimate. Think back to the earlier example of developing a web page. Now think about another web page also with 100 fields. But some are date fields with calendar widgets that pop up. Some are formatted text fields like phone numbers or Social Security numbers. This screen also requires interactions between fields. If the user enters a Visa card, a three-digit CVV field is shown. But if the user enters an American Express card, a four-digit CVV field is shown. Even though there are still 100 fields on this screen, these fields are harder to implement. They're more complex. They'll take more time. There's more chance the developer makes a mistake and has to back up and correct it. This additional complexity should be reflected in the estimate provided.

3. Risk & Uncertainty Involved: The amount of risk and uncertainty in a product backlog item should affect the story point estimate given to the item.



Summary

- In this lesson, you have learnt
  - Introduction to SCRUM
  - Different Scrum Roles and Responsibilities in Agile
  - Scrum Core Practices and Artifacts
  - Definition of "Done"
  - Estimation Techniques in Agile
  - Planning Poker®
  - Story Points

Add the notes here.

Introduction to Agile

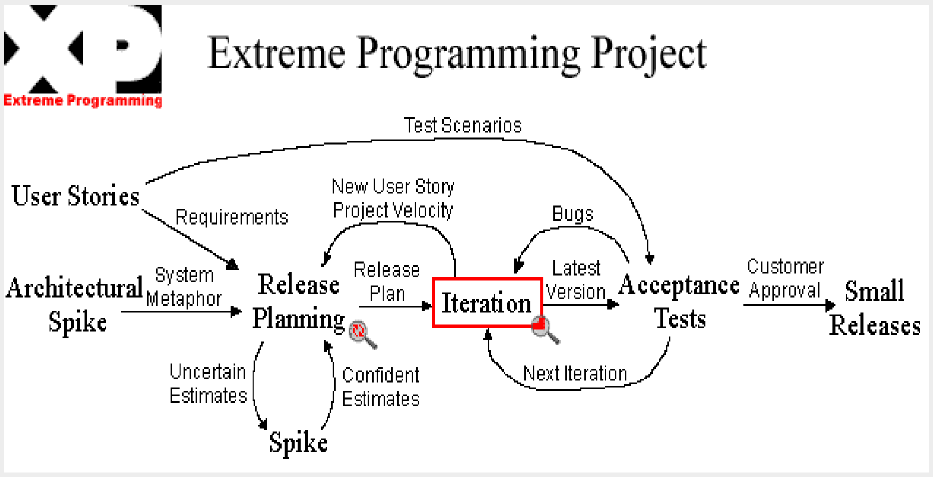
Lesson 3: Agile Methods and Practices - Extreme Programming (XP),Lean Software Development & Kanban

Lesson Objectives

- Introduction to Extreme Programming
- The Rules of Extreme Programming
- Extreme Programming (XP) - Principles
- Extreme Programming (XP) – Key Terms
- Introduction to Lean Software Development
- Principles of Lean Software Development
- What is Kanban?

3.1: Agile Methods and Practices – Extreme Programming (XP)

Introduction to Extreme Programming



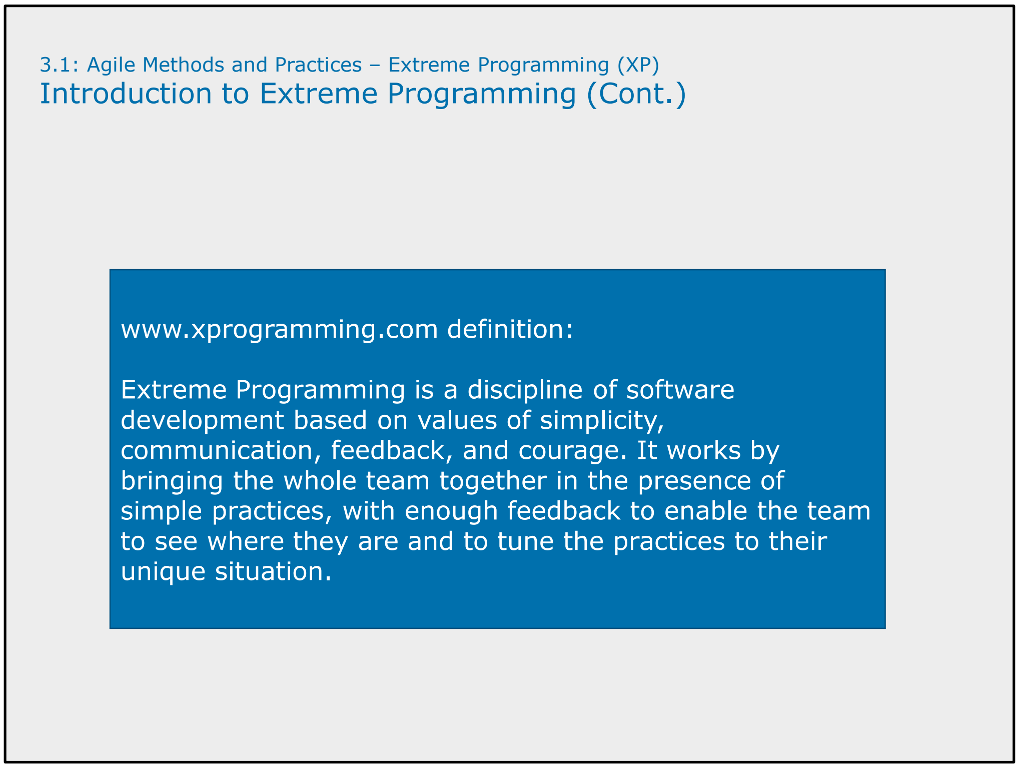
3.1: Agile Methods and Practices – Extreme Programming (XP)

Introduction to Extreme Programming (Cont.)

- Extreme programming (XP) was created by Kent Beck where he applied a light weight methodology to deliver a financial system in 2 years which previously had been undelivered over a number of years with a team of 30 people

Wikipedia definition:

Extreme Programming (XP) is a software development methodology which is intended to improve software quality and responsiveness to changing customer requirements. As a type of agile software development, it advocates frequent "releases" in short development cycles, which is intended to improve productivity and introduce checkpoints where new customer requirements can be adopted.



3.1: Agile Methods and Practices – Extreme Programming (XP)

Introduction to Extreme Programming (Cont.)

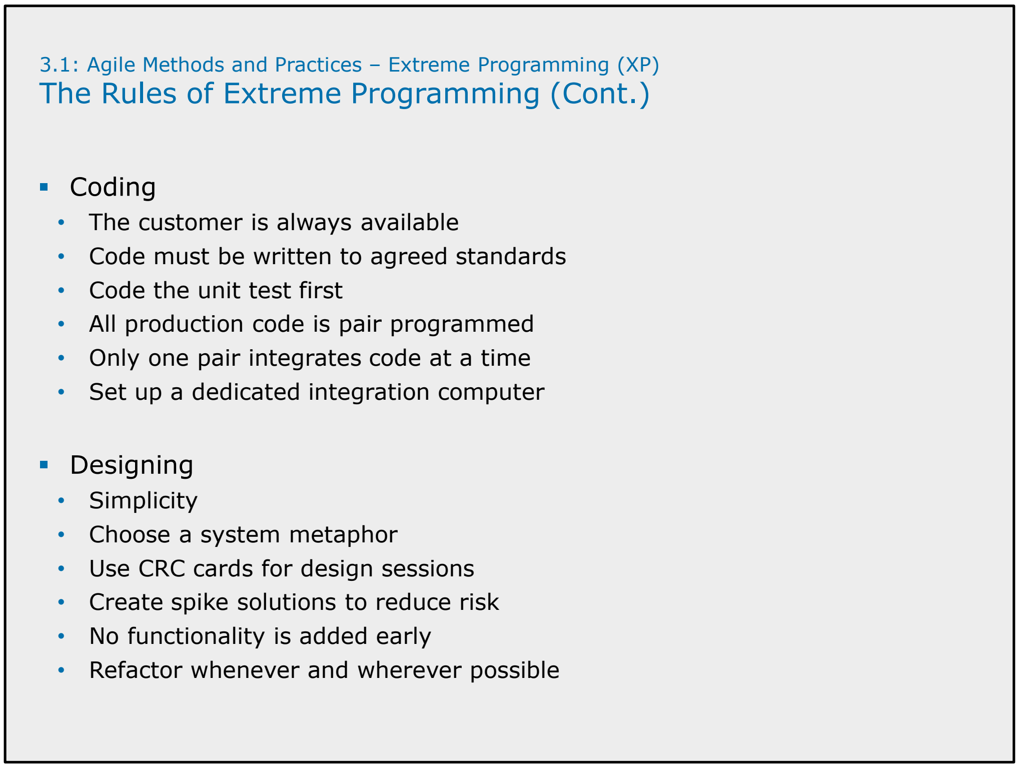
www.xprogramming.com definition:

Extreme Programming is a discipline of software development based on values of simplicity, communication, feedback, and courage. It works by bringing the whole team together in the presence of simple practices, with enough feedback to enable the team to see where they are and to tune the practices to their unique situation.

3.1: Agile Methods and Practices – Extreme Programming (XP)

The Rules of Extreme Programming

- Planning
  - User Stories User stories are written
  - Release planning creates the release schedule
  - Make frequent small releases
  - The project is divided into iterations
  - Iteration planning starts each iteration
- Managing
  - Give the team a dedicated open work space
  - Set a sustainable pace
  - A stand up meeting starts each day
  - The Project Velocity is measured
  - Move people around
  - Fix XP when it breaks



3.1: Agile Methods and Practices – Extreme Programming (XP)

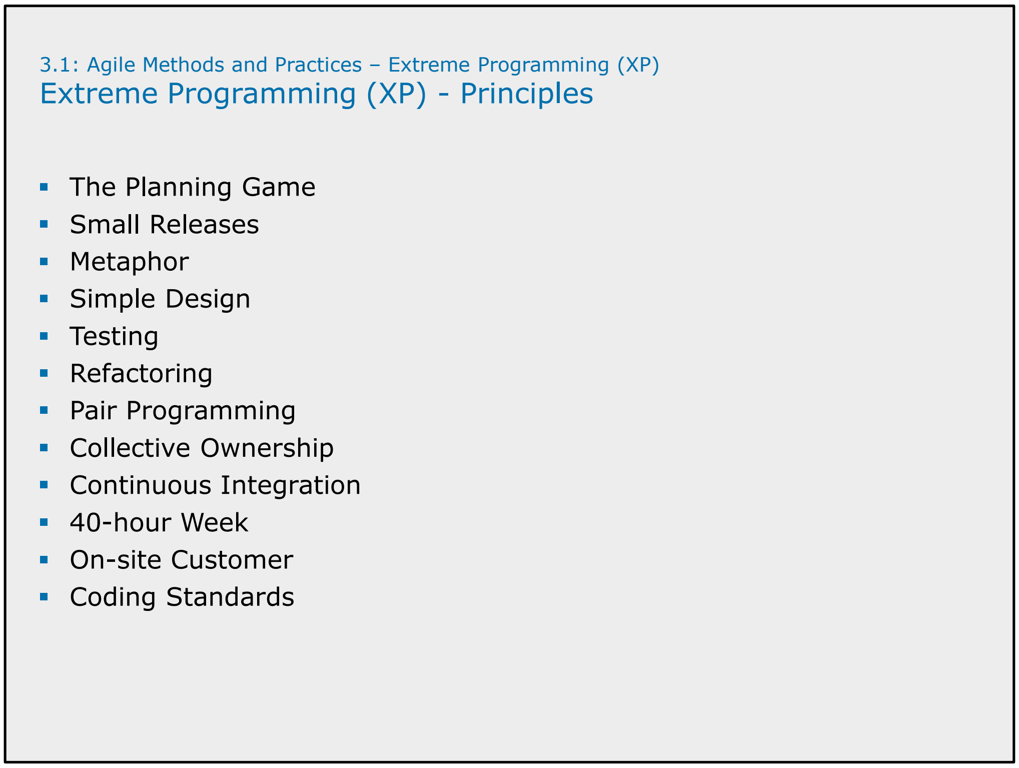
The Rules of Extreme Programming (Cont.)

- Coding
  - The customer is always available
  - Code must be written to agreed standards
  - Code the unit test first
  - All production code is pair programmed
  - Only one pair integrates code at a time
  - Set up a dedicated integration computer
- Designing
  - Simplicity
  - Choose a system metaphor
  - Use CRC cards for design sessions
  - Create spike solutions to reduce risk
  - No functionality is added early
  - Refactor whenever and wherever possible

3.1: Agile Methods and Practices – Extreme Programming (XP)

The Rules of Extreme Programming (Cont.)

- Testing
  - All code must have unit tests
  - All code must pass all unit tests before it can be released
  - When a bug is found tests are created.
  - Acceptance tests are run often and the score is published



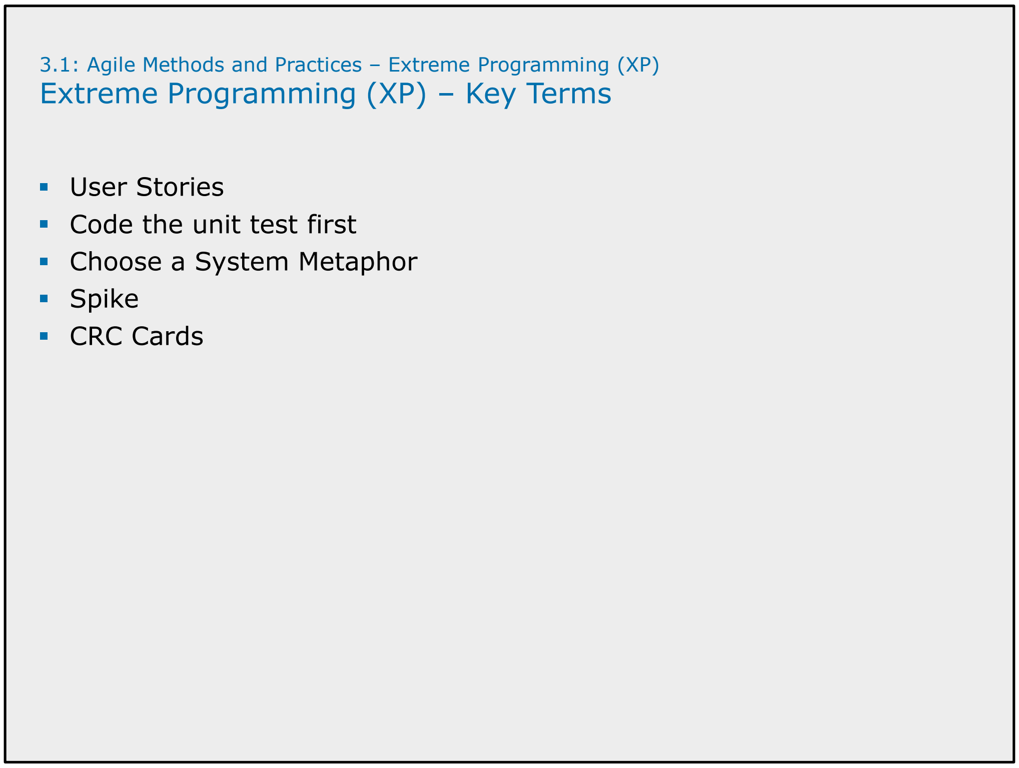
3.1: Agile Methods and Practices – Extreme Programming (XP)

Extreme Programming (XP) - Principles

- The Planning Game
- Small Releases
- Metaphor
- Simple Design
- Testing
- Refactoring
- Pair Programming
- Collective Ownership
- Continuous Integration
- 40-hour Week
- On-site Customer
- Coding Standards

Extreme Programming (XP) – Principles:

1. The Planning Game - Scope next release with business priorities, technical, estimates Update the plan based on reality.
2. Small Releases - Put simply system into production quickly Release new version in short cycle.
3. Metaphor - Guide development with simple shared story of how the whole system works.
4. Simple Design - Design as simple as possible at any given moment.
5. Testing - Continually write and run unit tests.
6. Refactoring - Restructure system without changing its behavior to remove duplication, improve communication, add flexibility and simplify.
7. Pair Programming - Two programmers, one machine, four eyes are better than two.
8. Collective Ownership - Anyone can change code anywhere in the system at any time.
9. Continuous Integration - Integrate and build the system many times a day, every time a talk is completed.
10. 40-hour Week - Never work overtime a second week in a row.
11. On-site Customer - Real, live user on the team, available full-time to answer questions.
12. Coding Standards - All code written accordance with rules emphasizing communication through the code.



3.1: Agile Methods and Practices – Extreme Programming (XP)

Extreme Programming (XP) – Key Terms

- User Stories
- Code the unit test first
- Choose a System Metaphor
- Spike
- CRC Cards

Extreme Programming (XP) – Key Terms

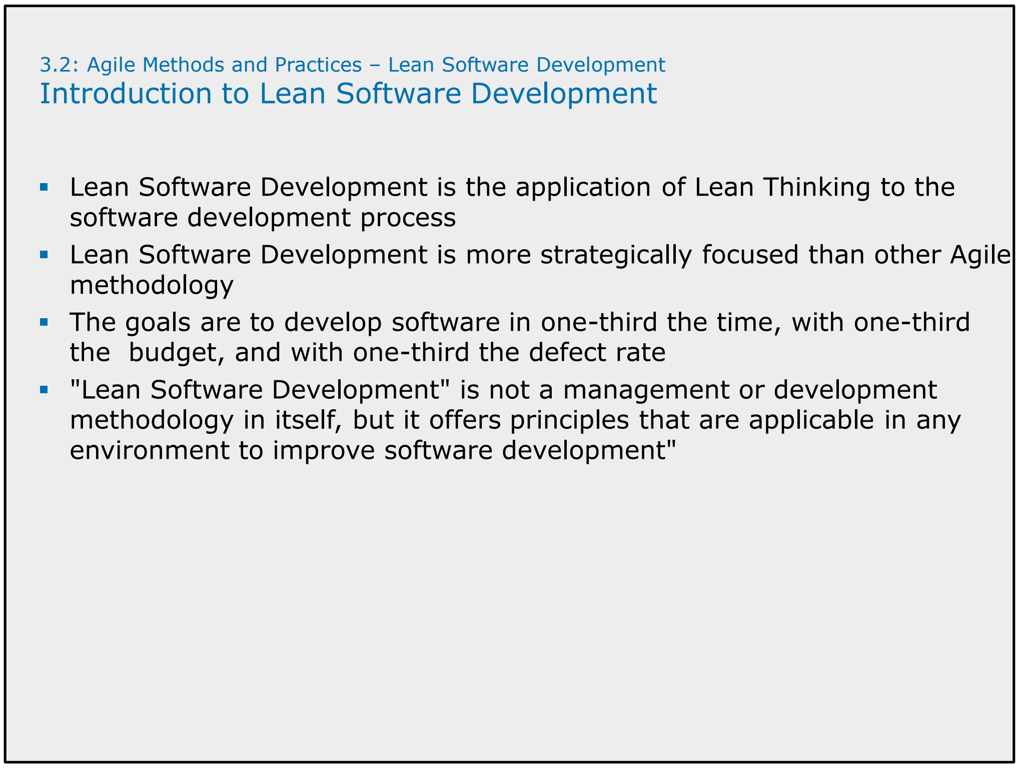
User Stories - User stories serve the same purpose as use cases but are not the same. They are used to create time estimates for the release planning meeting. They are also used instead of a large requirements document. User Stories are written by the customers as things that the system needs to do for them. They are similar to usage scenarios, except that they are not limited to describing a user interface. They are in the format of about three sentences of text written by the customer in the customers terminology without techno-syntax.

Code the unit test first –Test Driven Development (TDD) is a development technique that switches the coding approach from "code & test" to "test & code". By designing a test first you flush out design and testability issues earlier. Once you have your test, you write just enough code to make the test pass. You continue to iteratively and incrementally add new tests and new code to make the test pass.

Choose a System Metaphor - This is a story that everyone can tell to explain how the system works through the use of common and understandable terminology.

Spike - Create spike solutions to figure out answers to tough technical or design problems. A spike solution is a very simple program to explore potential solutions. Build the spike to only addresses the problem under examination and ignore all other concerns. Most spikes are not good enough to keep, so expect to throw it away. The goal is reducing the risk of a technical problem or increase the reliability of a user story's estimate. When a technical difficulty threatens to hold up the system's development put a pair of developers on the problem for a week or two and reduce the potential risk.

CRC Cards –Class, Responsibility and Collaboration cards are a design technique that uses cards to represent objects. You use a workshop based approach to then walk through how the objects interact with each other i.e. the messages that they send to other objects.



3.2: Agile Methods and Practices – Lean Software Development

Introduction to Lean Software Development

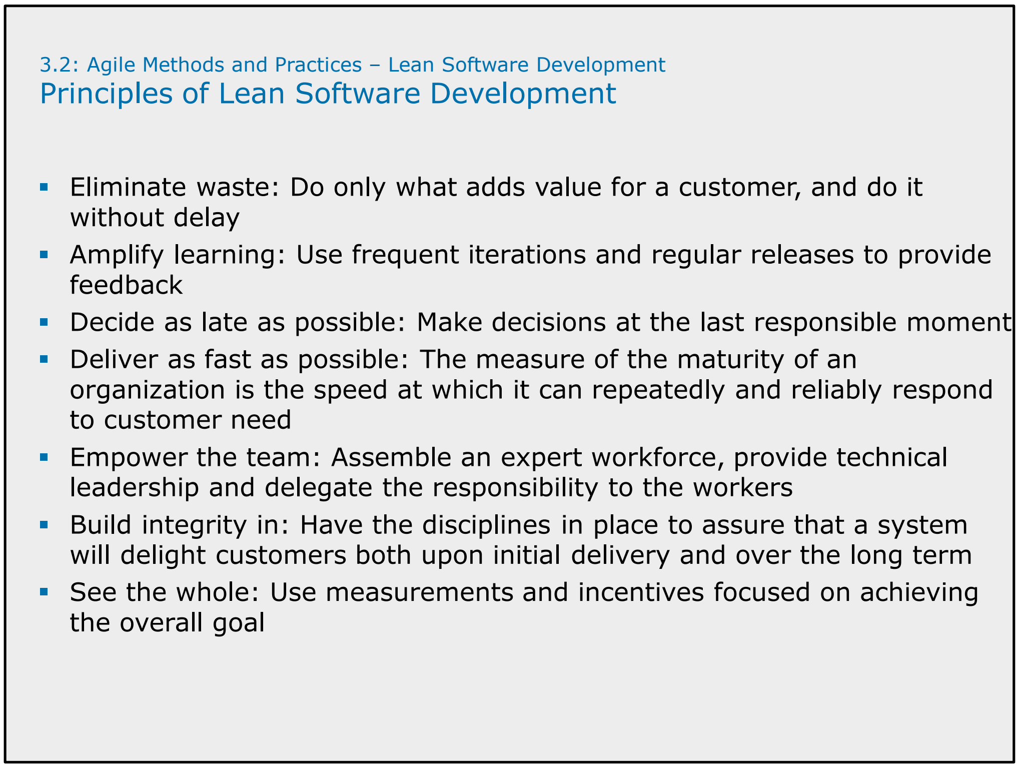
- Lean Software Development is the application of Lean Thinking to the software development process
- Lean Software Development is more strategically focused than other Agile methodology
- The goals are to develop software in one-third the time, with one-third the budget, and with one-third the defect rate
- "Lean Software Development" is not a management or development methodology in itself, but it offers principles that are applicable in any environment to improve software development"

What is Lean Software Development?

The concepts of a lean-enterprise originate from Toyota in Japan after the Second World War. Lean software development is a management philosophy that focuses on throughput. Lean software development doesn't focus on particular components of the value-stream like code-construction or QA, but on whether the components of the chain are working as efficiently as possible so as to generate as much value as possible to the customer.

Lean software development practices were created as a result of recognizing that enterprises were reacting to complex software development and resource challenges by adding:

- **More measurements**
- **More controls**
- **More checks and balances**
- **More process rigor**



3.2: Agile Methods and Practices – Lean Software Development

Principles of Lean Software Development

- Eliminate waste: Do only what adds value for a customer, and do it without delay
- Amplify learning: Use frequent iterations and regular releases to provide feedback
- Decide as late as possible: Make decisions at the last responsible moment
- Deliver as fast as possible: The measure of the maturity of an organization is the speed at which it can repeatedly and reliably respond to customer need
- Empower the team: Assemble an expert workforce, provide technical leadership and delegate the responsibility to the workers
- Build integrity in: Have the disciplines in place to assure that a system will delight customers both upon initial delivery and over the long term
- See the whole: Use measurements and incentives focused on achieving the overall goal

Principles of Lean Software Development

Eliminate waste - In software development, waste is anything that does not improve the quality of code, reduces the amount of time and effort it takes to produce code, or does not deliver business value to the customer. In other words, any activity that does not “pay for itself” in reduced effort elsewhere in the system.

Amplify Learning - For programmers to develop a system that delivers business value, they will have to learn about many things. Some are technical, such as the advantages and disadvantages to various approaches to do remote communications in .NET (i.e., remoting, COM+,web services, etc.). Others are requirements related, such as understanding what the business user really needs versus what the developer thinks the user needs.

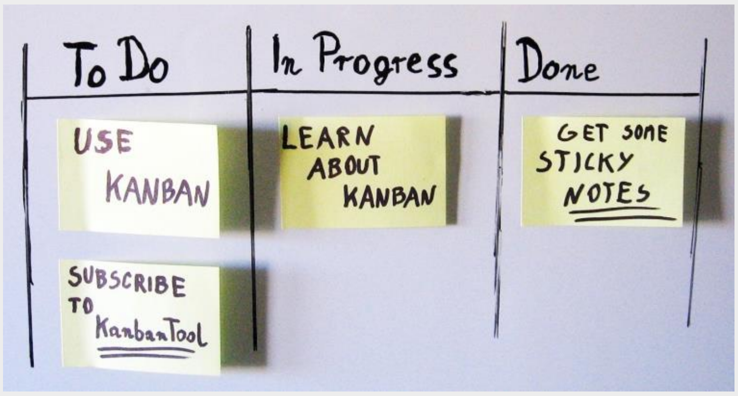
Decide as late as possible - The idea here is to wait until what the authors term “the last responsible moment” to make a decision. This is the moment at which, if the team does not make a decision, the decision will be made for them (doing nothing is a choice). The benefits of this are avoiding or delaying the costs of change, which obviously cannot be incurred if you have not limited your options yet.

Deliver as fast as possible - This is the foundation of iterative development. Requirements change as a percentage of the original requirements increases non-linearly as the amount of time increases. Typical 9-12 month projects generate roughly a 25 percent change in requirements. However, the amount of requirements change over a month averages only 1-2 percent. And it is much easier to get users to accept waiting until next month rather than next year.

3.2: Agile Methods and Practices – Kanban

What is Kanban?

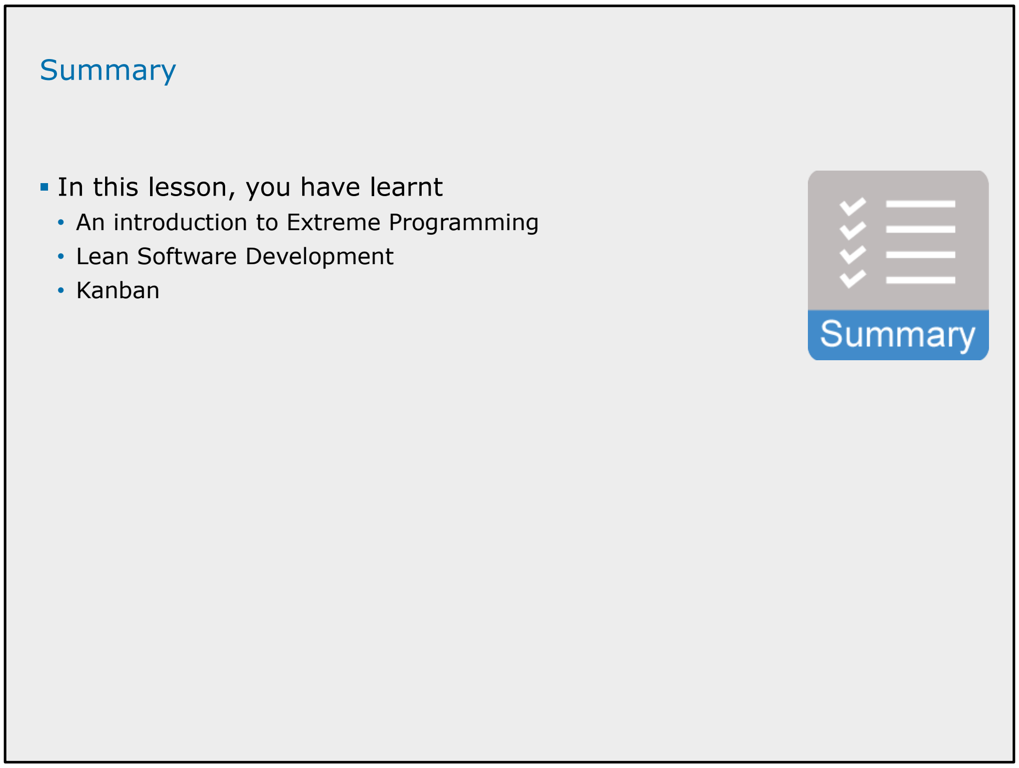
- The word Kan means "visual" in Japanese and the word "ban" means "card". So Kanban refers to "visual cards‟



3.2: Agile Methods and Practices – Kanban

What is Kanban? (Cont.)

- Kanban is way for teams and organizations to visualize their work, identify and eliminate bottlenecks and achieve dramatic operational improvements in terms of throughput and quality
- Kanban is a method to gradually improve whatever you do – whether software development, IT/ Ops, Staffing, Recruitment, Marketing and Sales
- in fact, almost any business function can benefit from applying Kanban to bring about significant benefits such as reduced lead time, increased throughput and much higher quality of products or services delivered



Summary

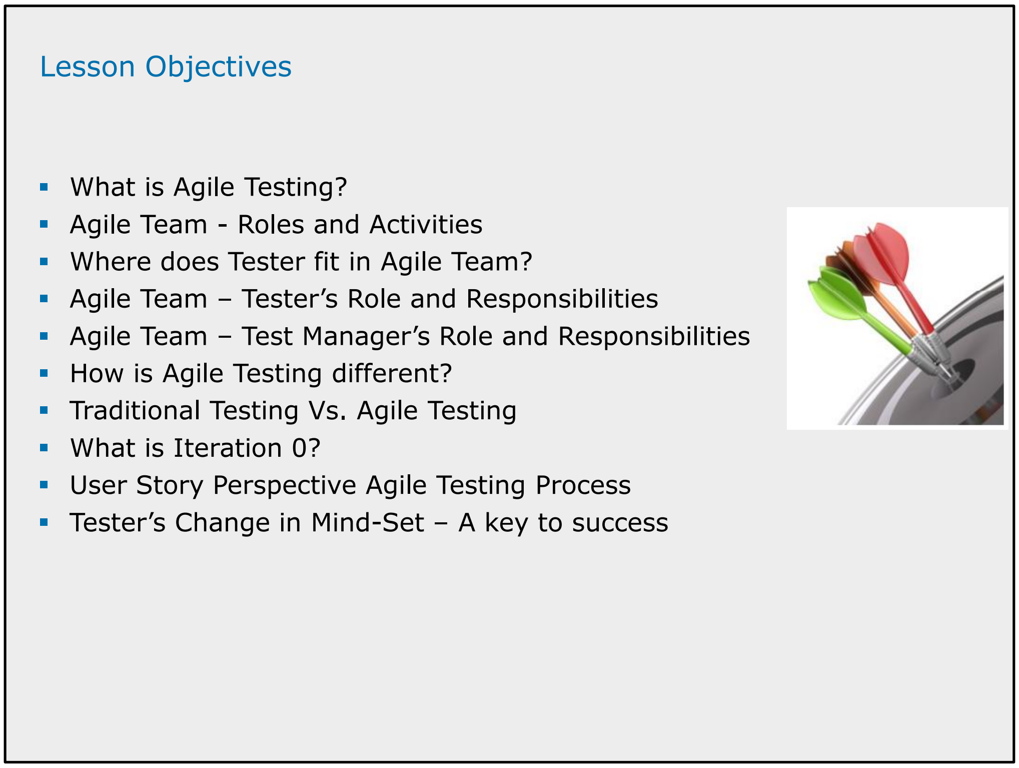
- In this lesson, you have learnt
  - An introduction to Extreme Programming
  - Lean Software Development
  - Kanban

Summary

Add the notes here.

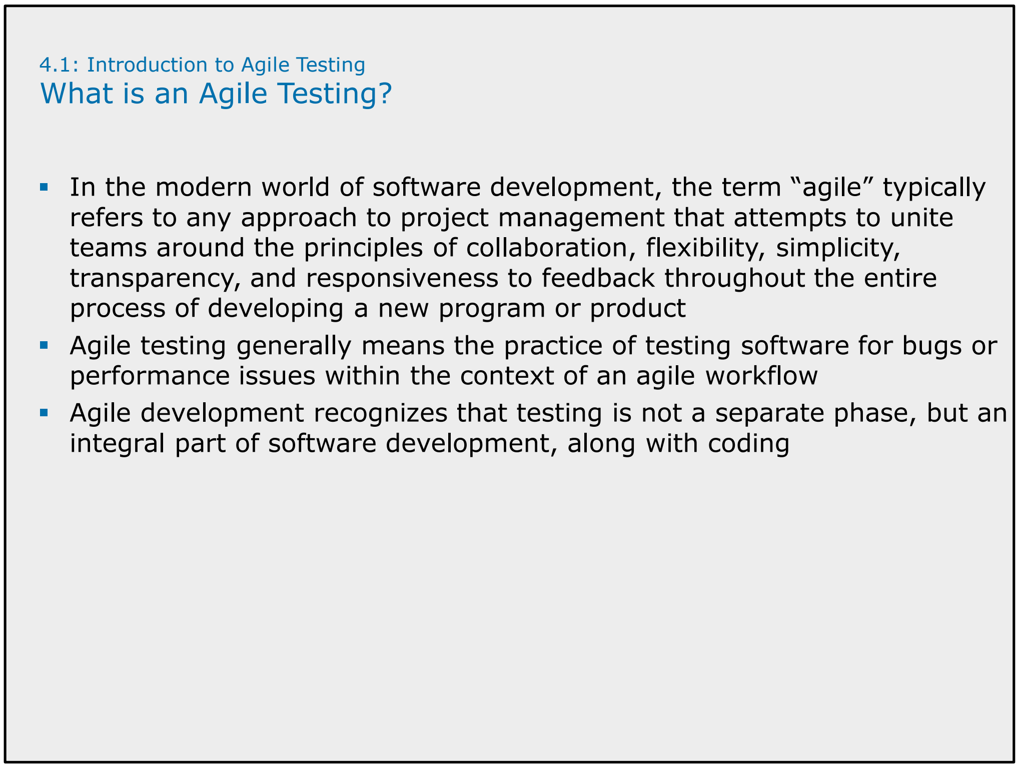
Introduction to Agile

Lesson 4: Introduction to Agile Testing



Lesson Objectives

- What is Agile Testing?
- Agile Team - Roles and Activities
- Where does Tester fit in Agile Team?
- Agile Team – Tester’s Role and Responsibilities
- Agile Team – Test Manager’s Role and Responsibilities
- How is Agile Testing different?
- Traditional Testing Vs. Agile Testing
- What is Iteration 0?
- User Story Perspective Agile Testing Process
- Tester’s Change in Mind-Set – A key to success



4.1: Introduction to Agile Testing

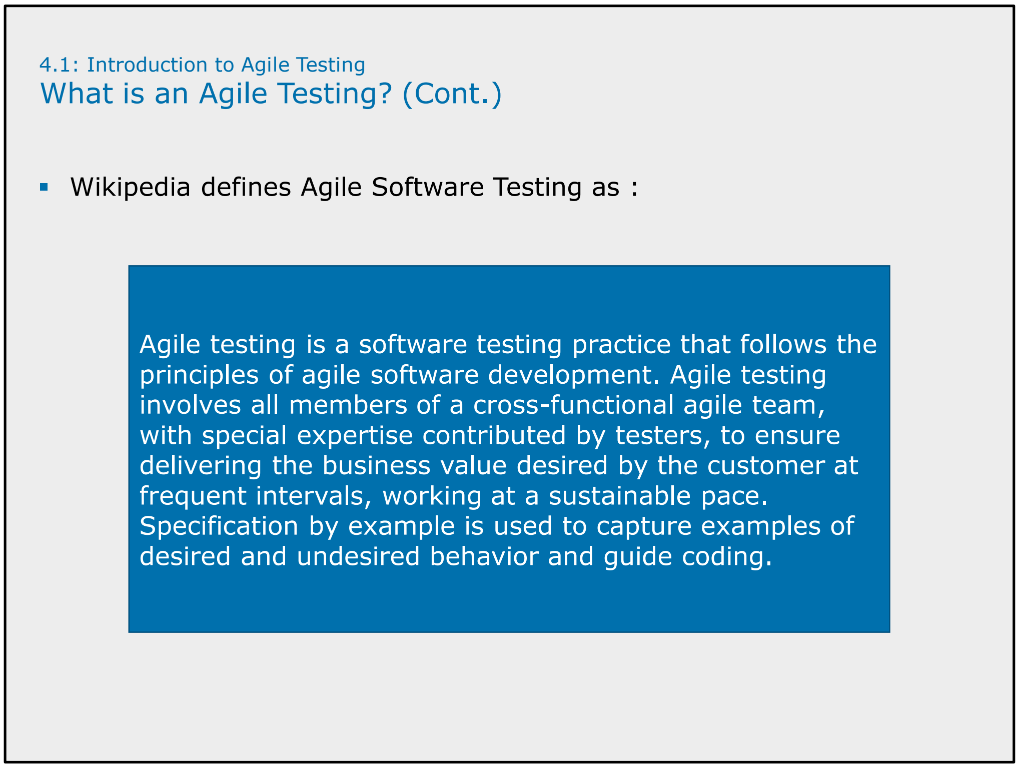
What is an Agile Testing?

- In the modern world of software development, the term “agile” typically refers to any approach to project management that attempts to unite teams around the principles of collaboration, flexibility, simplicity, transparency, and responsiveness to feedback throughout the entire process of developing a new program or product
- Agile testing generally means the practice of testing software for bugs or performance issues within the context of an agile workflow
- Agile development recognizes that testing is not a separate phase, but an integral part of software development, along with coding

What is Agile Testing?

In the modern world of software development, the term “agile” typically refers to any approach to project management that attempts to unite teams around the principles of collaboration, flexibility, simplicity, transparency, and responsiveness to feedback throughout the entire process of developing a new program or product. Agile testing generally means the practice of testing software for bugs or performance issues within the context of an agile workflow.

Agile development identifies that testing is not a separate phase, but an it is an integral part of software development, alongside coding. Testers on agile teams lend their expertise in eliciting examples of desired behavior from customers, collaborating with the development team to turn those into executable specifications that guide coding. Testing and coding are done incrementally and iteratively, building up each feature until it provides enough value to release to production.
Agile testing covers all types of testing. The Agile Testing Quadrants provide a helpful taxonomy to help teams identify and plan the testing needed.
Agile Testing doesn’t just mean testing on an agile project. Some testing approaches such as exploratory testing, are inherently agile, whether its done on an agile project or not.

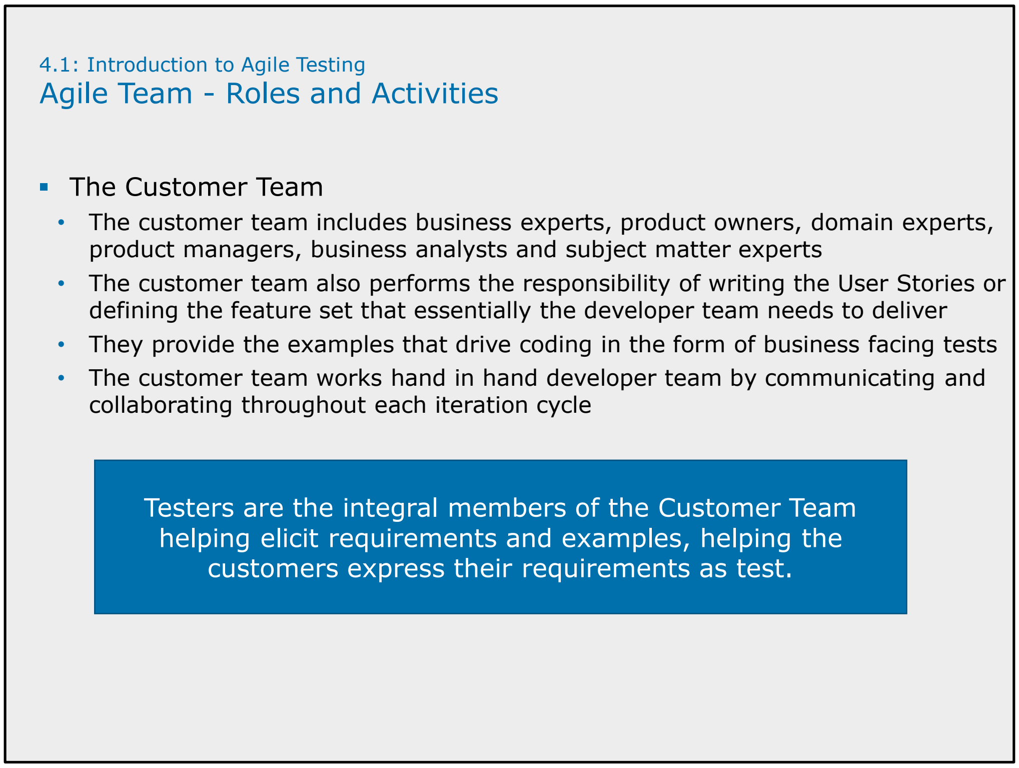


4.1: Introduction to Agile Testing

What is an Agile Testing? (Cont.)

- Wikipedia defines Agile Software Testing as :

Agile testing is a software testing practice that follows the principles of agile software development. Agile testing involves all members of a cross-functional agile team, with special expertise contributed by testers, to ensure delivering the business value desired by the customer at frequent intervals, working at a sustainable pace. Specification by example is used to capture examples of desired and undesired behavior and guide coding.



4.1: Introduction to Agile Testing

Agile Team - Roles and Activities

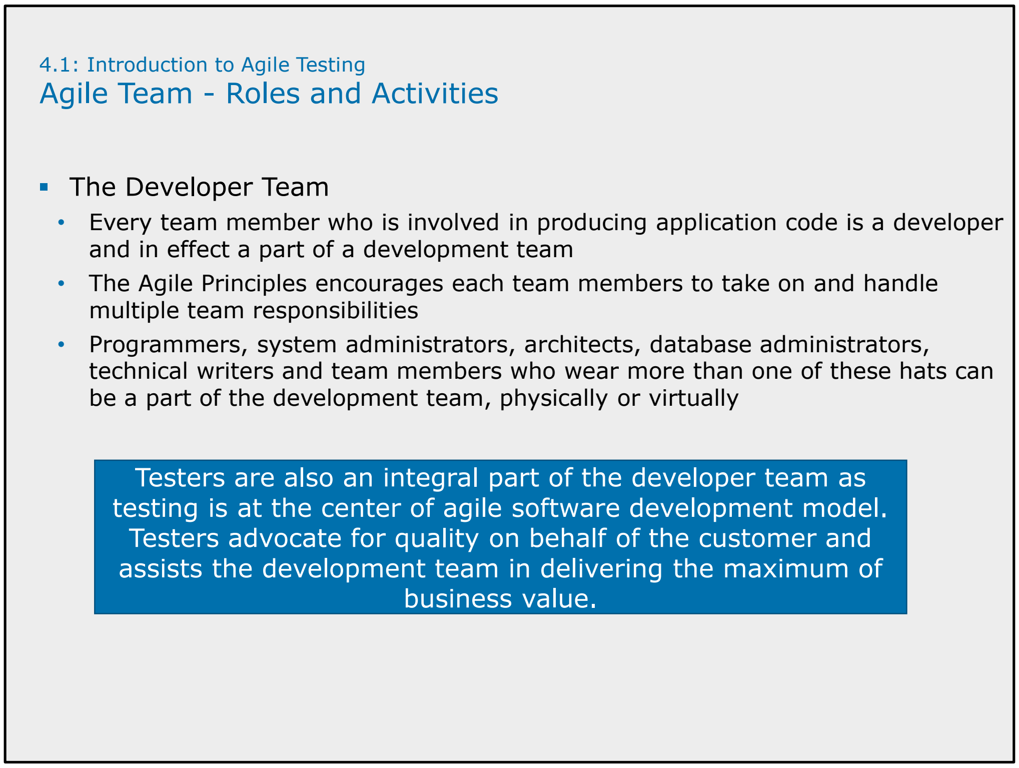
- The Customer Team
  - The customer team includes business experts, product owners, domain experts, product managers, business analysts and subject matter experts
  - The customer team also performs the responsibility of writing the User Stories or defining the feature set that essentially the developer team needs to deliver
  - They provide the examples that drive coding in the form of business facing tests
  - The customer team works hand in hand developer team by communicating and collaborating throughout each iteration cycle

Testers are the integral members of the Customer Team helping elicit requirements and examples, helping the customers express their requirements as test.

Agile Team – Roles and Activities

While there is no "magical" structure for an Agile team, it is important to understand what the roles of each of the key members of the team are.

The Customer Team - The customer team includes business experts, product owners, domain experts, product managers, business analysts and subject matter experts. The customer team also performs the responsibility of writing the User Stories or defining the feature set that essentially the developer team needs to deliver. They provide the examples that drive coding in the form of business facing tests. The customer team works hand in hand developer team by communicating and collaborating throughout each iteration cycle. They use various effective techniques to do so i.e. by answering questions, drawing examples on the whiteboard and reviewing finished stories or the parts of the stories.



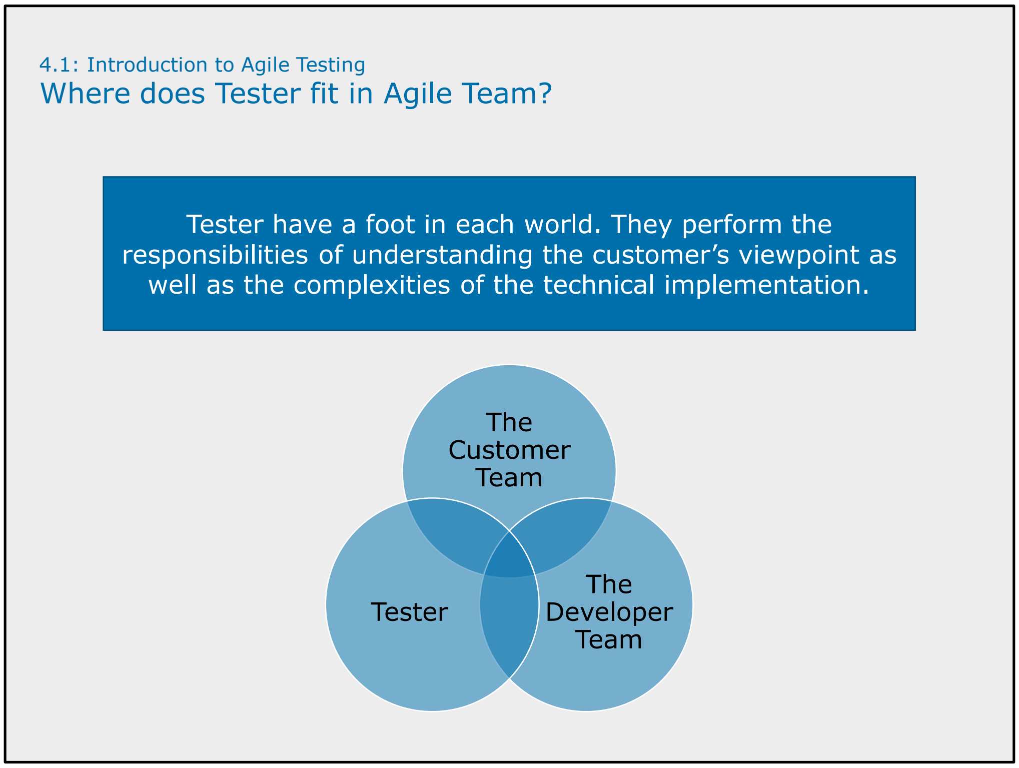
4.1: Introduction to Agile Testing

Agile Team - Roles and Activities

- The Developer Team
  - Every team member who is involved in producing application code is a developer and in effect a part of a development team
  - The Agile Principles encourages each team members to take on and handle multiple team responsibilities
  - Programmers, system administrators, architects, database administrators, technical writers and team members who wear more than one of these hats can be a part of the development team, physically or virtually

Testers are also an integral part of the developer team as testing is at the center of agile software development model. Testers advocate for quality on behalf of the customer and assists the development team in delivering the maximum of business value.

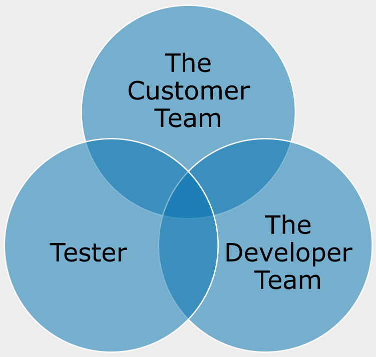
The Developer Team - Every team member who is involved in producing application code is a developer and in effect a part of a development team. The Agile Principles encourages each team members to take on and handle multiple team responsibilities. Many agile practitioners discourage specialized roles on agile team and encourage all team members to transfer their skills to others as much as possible. Programmers, system administrators, architects, database administrators, technical writers and team members who wear more than one of these hats can be a part of the development team, physically or virtually.



4.1: Introduction to Agile Testing

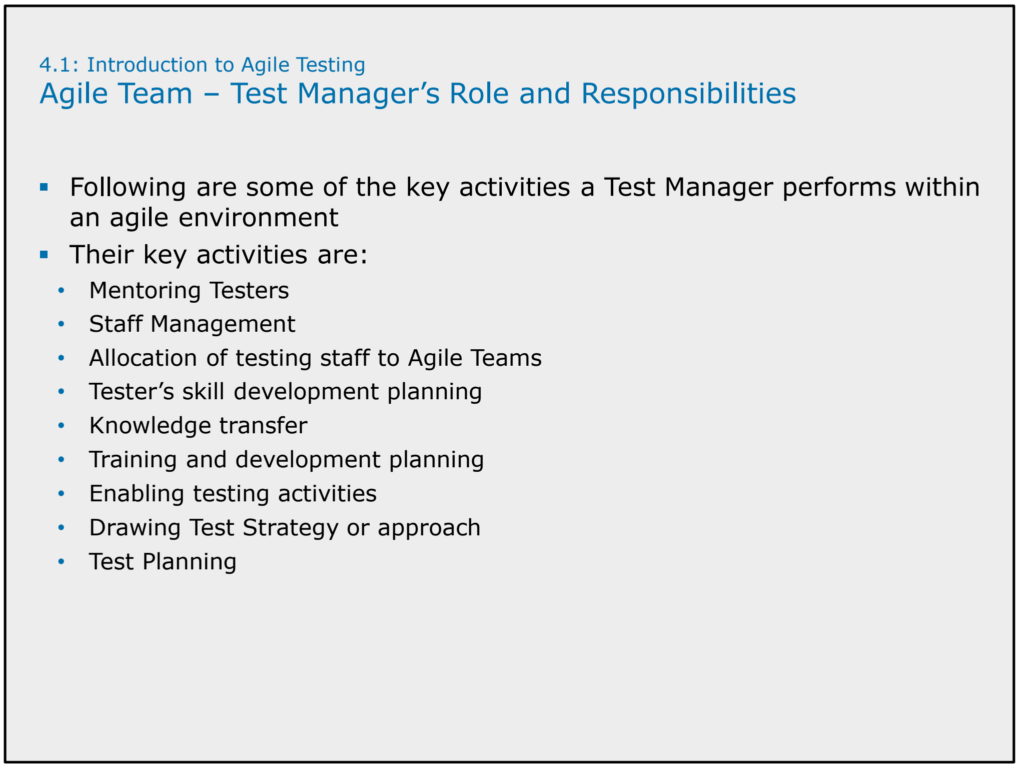
Where does Tester fit in Agile Team?

Tester have a foot in each world. They perform the responsibilities of understanding the customer's viewpoint as well as the complexities of the technical implementation.



Where does Tester fit in Agile Team?

Sometimes some agile teams don't have any member in particular who define themselves as testers. However, they all need someone who can assist the customer team in writing functionalities facing test for user stories of each iteration, ensure that the tests pass, and make sure that adequate regression tests are automated. Even under the situation when the team does have testers, the whole agile team would be responsible for the testing of iteration user stories. The various agile project experiences talks the stories where testing skills and experiences are very much important to project success and that testers do add value to agile teams.



4.1: Introduction to Agile Testing

Agile Team – Test Manager’s Role and Responsibilities

- Following are some of the key activities a Test Manager performs within an agile environment
- Their key activities are:
  - Mentoring Testers
  - Staff Management
  - Allocation of testing staff to Agile Teams
  - Tester’s skill development planning
  - Knowledge transfer
  - Training and development planning
  - Enabling testing activities
  - Drawing Test Strategy or approach
  - Test Planning

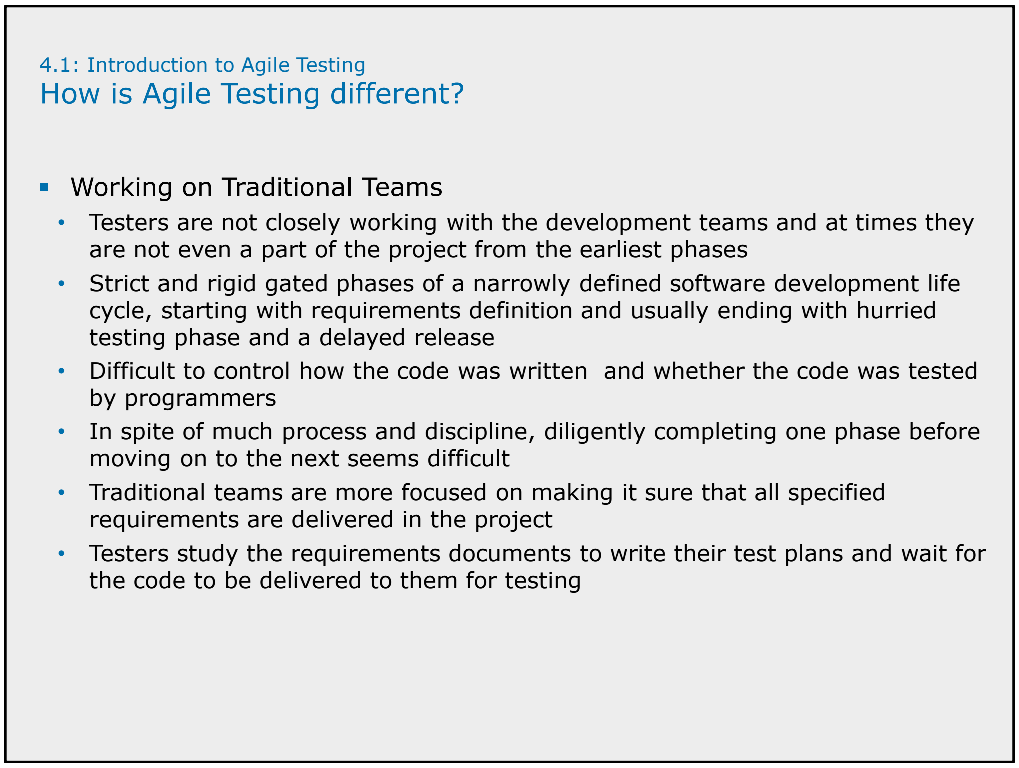
Agile Team – Test Manager’s Role and Responsibilities

The role and the responsibilities of a traditional test manager changes when organization decides to move to Agile. In some cases this may cause issues as they will no longer be the "manager" as the Agile team itself needs to be self managing. They play the role of a staff manager rather than of a manager. They will provide time to time support to Scrum Master and Project Manager, if there is still one, in the overall project delivery.
They will also mentor the testers by providing guidance on skills and competencies required by the testers in Agile environment. The test manager should be mentoring the testers to be self-managed. They may support the activity of formation of the Agile team.
Test Managers will also facilitate, along with the Scrum Master, the management and removal of roadblocks. In some instances the Test Manager may anticipate a roadblock and ensure that it is removed before the team identifies it. Their experience can also be used during the risk identification and analysis during the initial reviews of the user stories. This of course can be used as input into assessing the technical difficulty of implementation of the user stories during the estimation period. They may also work with the team on the Test Strategy/Approach during the planning meetings at the Release and Iteration levels.

4.1: Introduction to Agile Testing

Agile Team – Tester’s Role and Responsibilities

- Testers are the important part of an Agile team
  - They start writing test cases even before code starts to be created by reviewing the user stories
  - Every story that reaches the iteration boundary i.e. the end of one iteration and the start of the next) is reviewed and tested
  - Their key responsibilities are:
  - Working with programmers and product owners to ensure that the stories are clearly understood
  - Ensuring that the acceptance tests track the desired functionality of the story
  - Creating the acceptance tests while the code is being created
  - Executing the acceptance tests against the code of the story
  - Ensuring that the test cases are checked into the version control system every day
  - Writing and maintaining automated tests that can be executed, as part of the continuous integration and testing process, against the code every day



4.1: Introduction to Agile Testing

How is Agile Testing different?

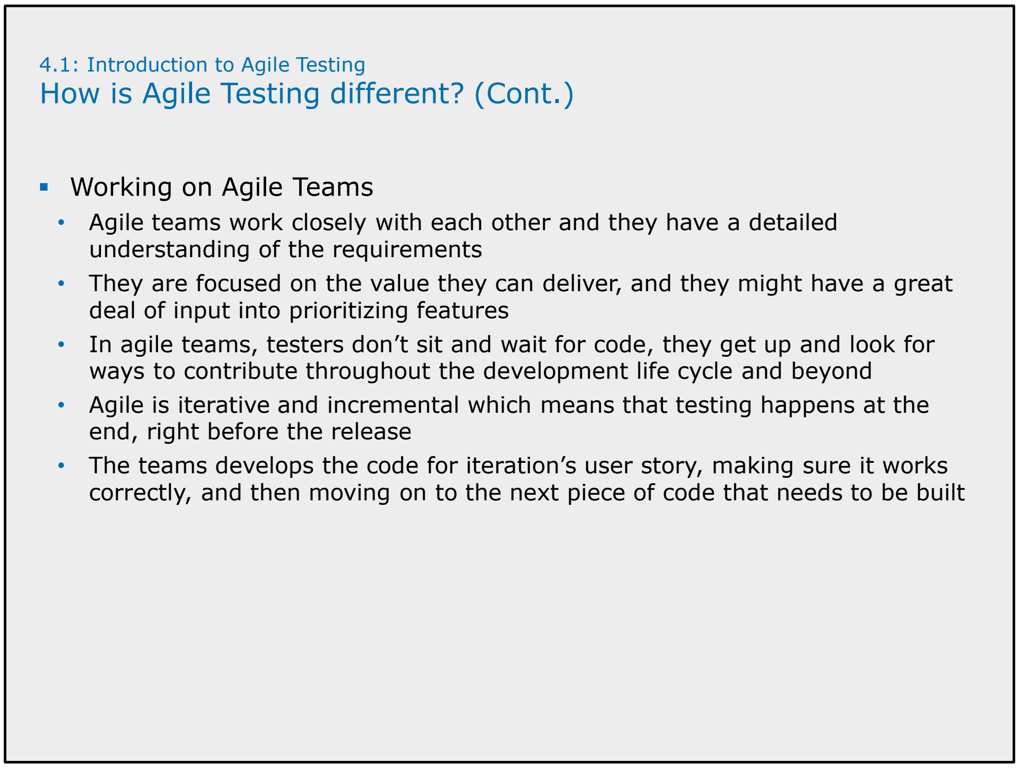
- Working on Traditional Teams
  - Testers are not closely working with the development teams and at times they are not even a part of the project from the earliest phases
  - Strict and rigid gated phases of a narrowly defined software development life cycle, starting with requirements definition and usually ending with hurried testing phase and a delayed release
  - Difficult to control how the code was written and whether the code was tested by programmers
  - In spite of much process and discipline, diligently completing one phase before moving on to the next seems difficult
  - Traditional teams are more focused on making it sure that all specified requirements are delivered in the project
  - Testers study the requirements documents to write their test plans and wait for the code to be delivered to them for testing

How is Agile Team different?

Working on Traditional Teams

As a part of traditional teams, testers, neither they are working closely with the programmers nor they are getting an opportunity to be a part of the project from its initial phases. The traditional approach of software development comprises of much stricter and rigid gated phases of a narrowly defined software development life cycle, starting with requirement definition and usually ending with rushed testing phase and delayed release. On traditional teams of software development, testers couldn’t control how the code was written and or even if any programmers tested their code.

In spite of much process and discipline, it is difficult to diligently completing one phase before moving on to the next phase. Traditional teams are more focused on making sure all the system requirements are delivered in the final product. If everything is not ready by the targeted release date, then the release is usually postponed. Testers study the requirement documents to write their test plans and wait for the code to be delivered to them before testing.



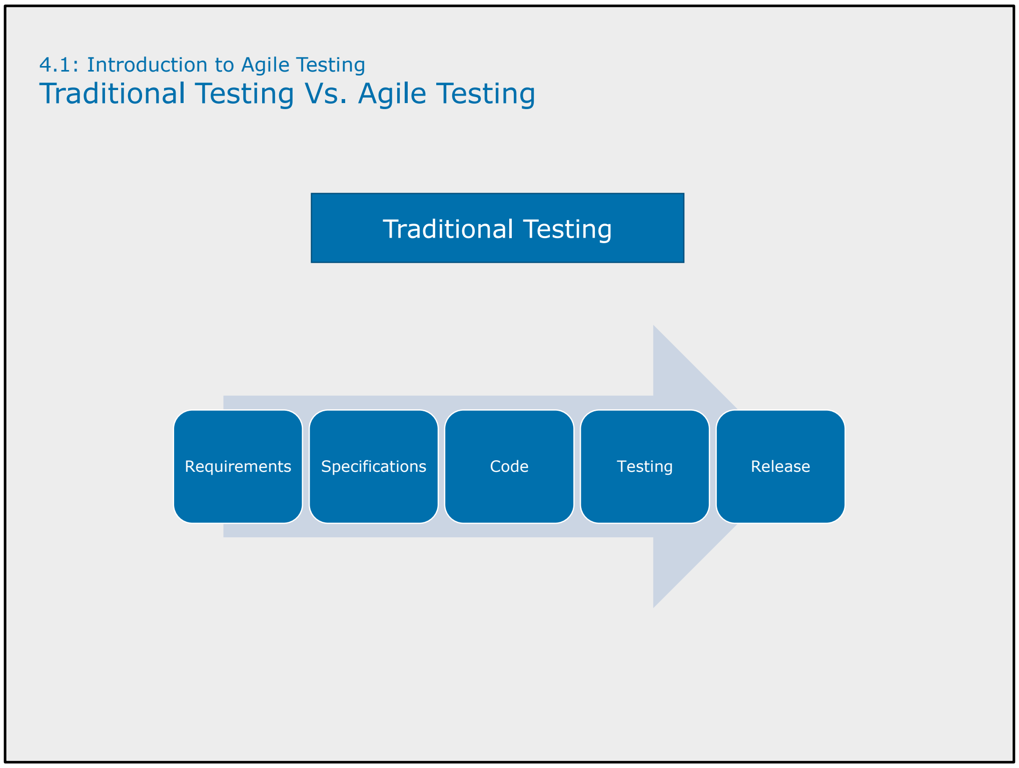
4.1: Introduction to Agile Testing

How is Agile Testing different? (Cont.)

- Working on Agile Teams
  - Agile teams work closely with each other and they have a detailed understanding of the requirements
  - They are focused on the value they can deliver, and they might have a great deal of input into prioritizing features
  - In agile teams, testers don't sit and wait for code, they get up and look for ways to contribute throughout the development life cycle and beyond
  - Agile is iterative and incremental which means that testing happens at the end, right before the release
  - The teams develops the code for iteration's user story, making sure it works correctly, and then moving on to the next piece of code that needs to be built

Working on Agile Teams

Agile teams work closely with each other and they have a detailed understanding of the requirements. They are focused on the value they can deliver, and they might have a great deal of input into prioritizing features. In agile teams, testers don't sit and wait for code, they get up and look for ways to contribute throughout the development life cycle and beyond. Agile is iterative and incremental which means that testing happens at the end, right before the release. The teams develops the code for iteration's user story, making sure it works correctly, and then moving on to the next piece of code that needs to be built.

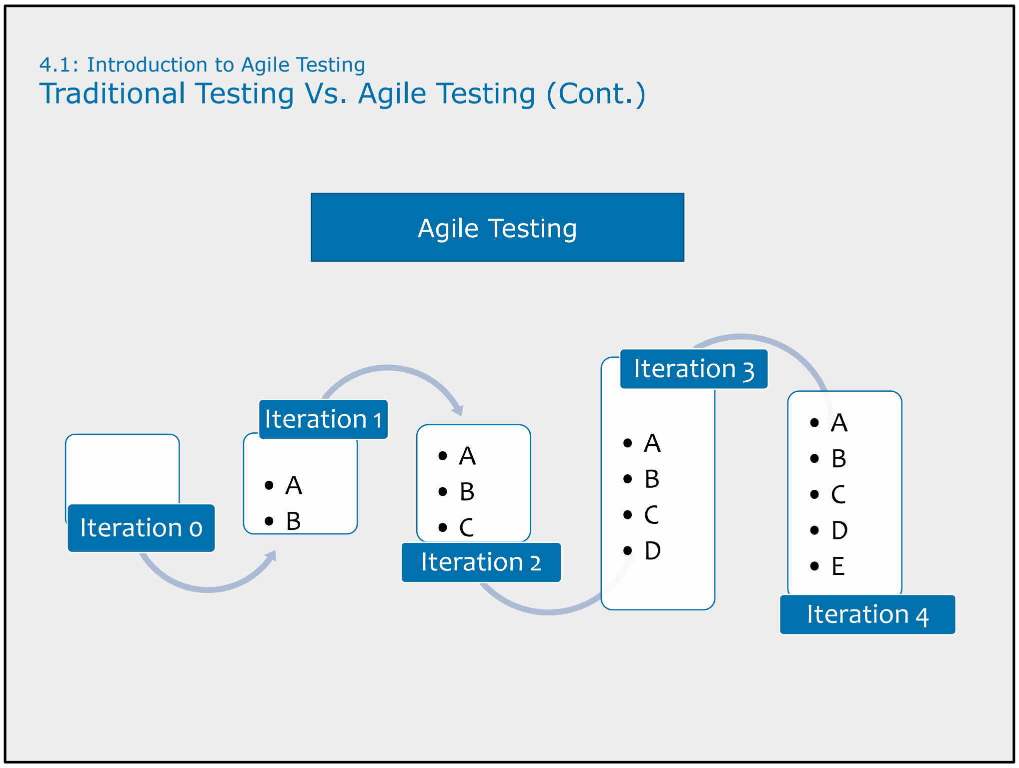


4.1: Introduction to Agile Testing

Traditional Testing Vs. Agile Testing

Traditional Testing

From the traditional testing diagram it clear that testing takes place towards the end of the software development life cycle and before the product release. The process looks idealistic because it gives an impression that there is as much time for testing as there is for coding. In many projects this is not the case. The gets squeezed because coding takes longer than expected and because teams get into the code-and-fix cycle at the end.



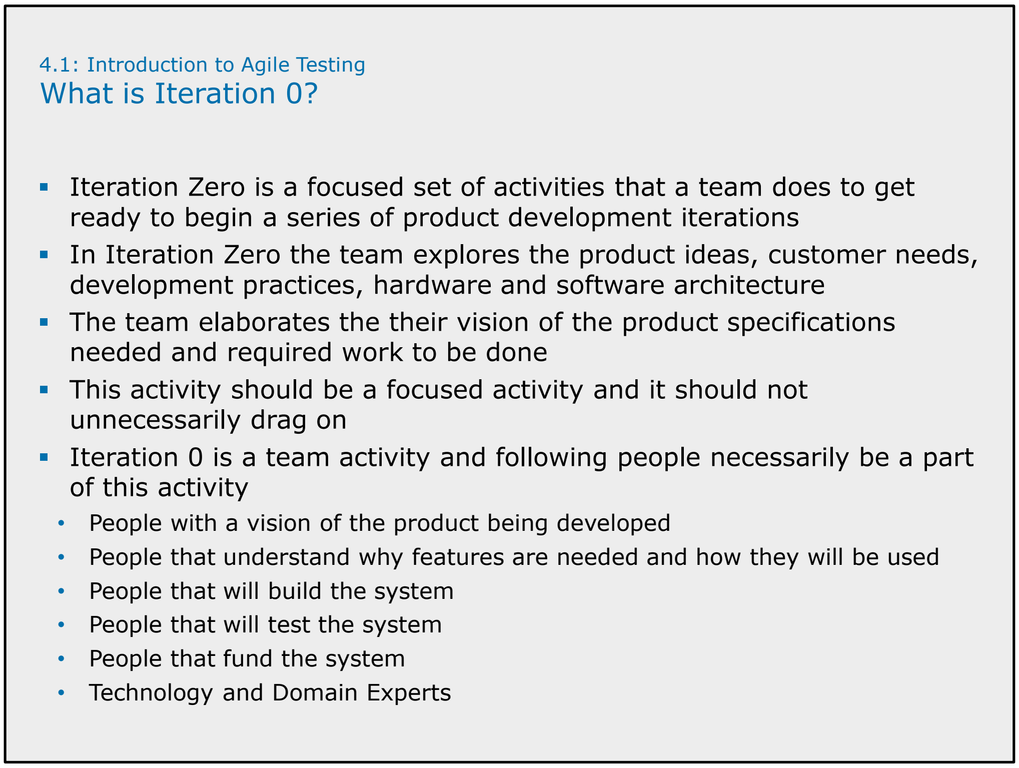
4.1: Introduction to Agile Testing

Traditional Testing Vs. Agile Testing (Cont.)

Agile Testing

Agile is iterative and incremental which means that the testers test each increment to the coding as soon as it is finished. The iteration might be as short as one week or as long as one month. The development team develops the solution for the iteration's user story, ensuring it works as expected and then moving on to the next piece of the code that needs to be built. On Agile teams developers never gets ahead of the testers, because the story is not "Completed" until it has been tested.

As a Tester on an Agile team, is a key player in releasing code to production. The most important difference for a tester in agile testing is the quick feedback from the testing. It drives the project forward, and there are no gatekeepers ready to block project progress if certain milestones are not met.



4.1: Introduction to Agile Testing

What is Iteration 0?

- Iteration Zero is a focused set of activities that a team does to get ready to begin a series of product development iterations
- In Iteration Zero the team explores the product ideas, customer needs, development practices, hardware and software architecture
- The team elaborates the their vision of the product specifications needed and required work to be done
- This activity should be a focused activity and it should not unnecessarily drag on
- Iteration 0 is a team activity and following people necessarily be a part of this activity
  - People with a vision of the product being developed
  - People that understand why features are needed and how they will be used
  - People that will build the system
  - People that will test the system
  - People that fund the system
  - Technology and Domain Experts

What is Iteration 0?

There seems to be lot of discussion around whether there should or shouldn’t be an iteration 0. Many of the Agile perfectionists strongly believe that there should be some working functionality that should be delivered after each iteration. For the Agile projects those who have iteration o it is generally considered as “Warm Up” or “Project Initiation” period.

The Iteration 0 deliverables:

1. Actively working with the stakeholders to either help them understand Agile and where they fit into it or to help them model the initial scope of the system
2. Create some high level prototypes
3. Start to build the team
4. Model the initial architecture
5. Set up the environment
6. Decide on the definition of "Done"
7. Select and configure any tools that are required
8. Decide on which methodology or blend of them to use

It is possible that what is decided in iteration 0 may not be what ultimately happens, but the team has to start somewhere**.**

4.1: Introduction to Agile Testing

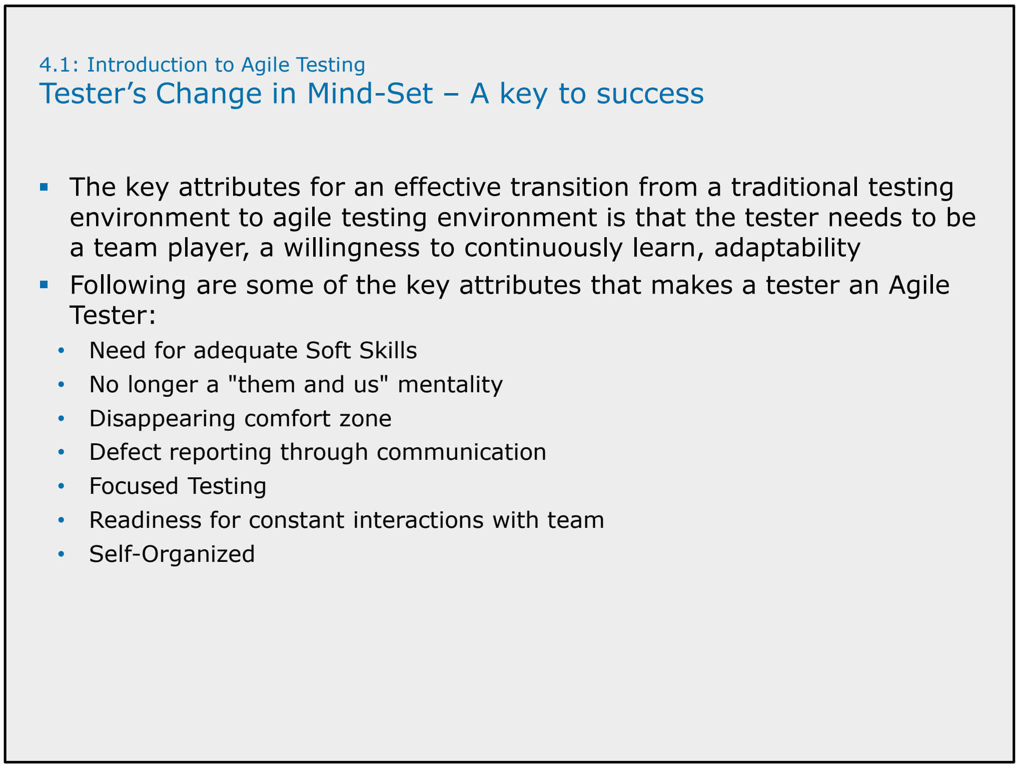
What is Iteration 0? (Cont.)

- The factors that can influence whether there should or should not be an Iteration Zero
  - The maturity of the team
  - Whether stakeholders have worked on an Agile project before
  - The quality of the user stories or understanding of the requirements that are being developed
  - The environments being used
  - Whether the environments are already setup
  - The level of experience the organization has with agile methods

4.1: Introduction to Agile Testing

User Story Perspective Agile Testing Process

- The test cases are written even before the coding begins that illustrates requirement for each user story rather than creating tests from requirement document
- This is often a combined effort between a domain expert and a tester, analyst or some other development team member
- Ideally based on examples provided by business experts detailed functional test cases are written
- Testers can also conduct manual exploratory testing to find important bugs that define test cases might miss on
- Testers might pair with other developers to automate and execute test cases as coding on each story proceeds
- Automated functional tests are added to the regression test suite
- When test demonstration minimum functionality are complete, the team can consider the story finished



4.1: Introduction to Agile Testing

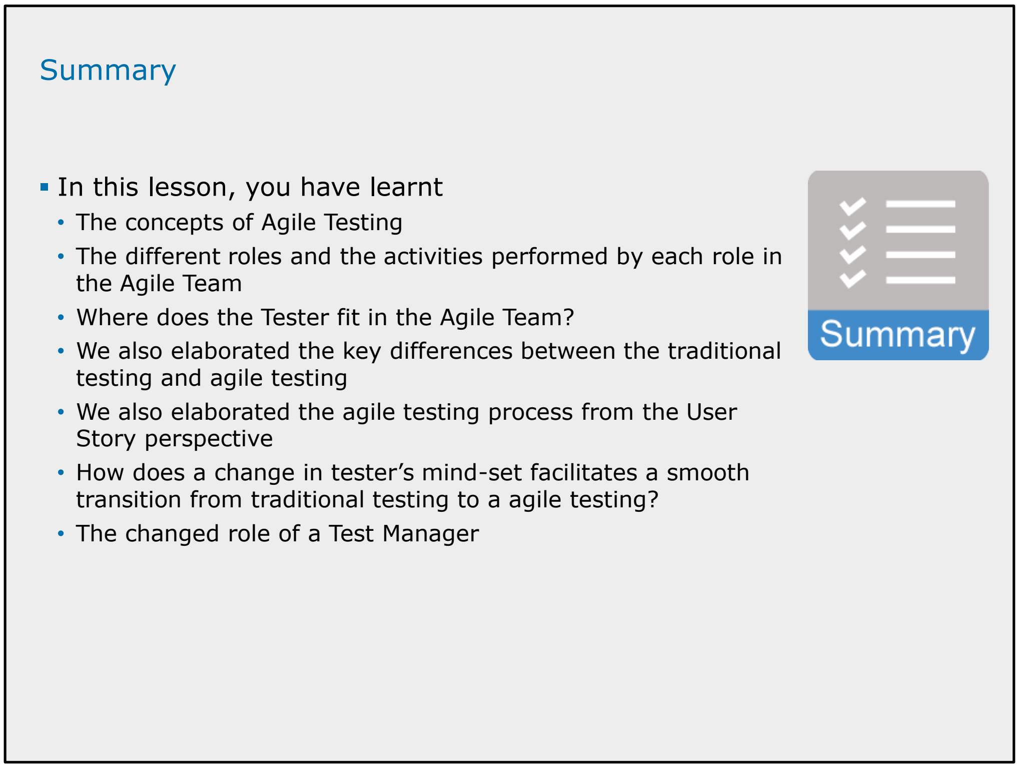
Tester’s Change in Mind-Set – A key to success

- The key attributes for an effective transition from a traditional testing environment to agile testing environment is that the tester needs to be a team player, a willingness to continuously learn, adaptability
- Following are some of the key attributes that makes a tester an Agile Tester:
  - Need for adequate Soft Skills
  - No longer a "them and us" mentality
  - Disappearing comfort zone
  - Defect reporting through communication
  - Focused Testing
  - Readiness for constant interactions with team
  - Self-Organized

Tester’s Change in Mind-Set – A key to success

A successful transition for a tester into an agile project is a change in their mindset. Many of the skills and competencies that they will already have can be used equally effectively in an agile environment however they need to be aware of how this environment may require them to use these in a slightly different way.

1. **Need for adequate Soft Skills** – In Agile Testing, tester’s soft skills plays a very important role as there is likely to be higher and continuous communication amongst the team. Testers need to be able to converse from a technical perspective with the developers and for automation. Also, they should be able to talk in the language the customer will understand.
2. **No longer a "them and us" mentality** - There is no longer a "them and us" mentality and it is the teams responsibility to ensure that user stories get implemented correctly and tested correctly in order to produce business outcome.
3. **Disappearing comfort zone** - Testers may be taken out of their comfort zone, where in a traditional project they may have had weeks if not months to plan and prepare their testing. Now they have only few days. They may have been used of using documentation with all the details written down for them however there will be less documentation in an agile project and it is the testers skill which will need to elicit the additional detail required.
4. **Defect reporting through communication** – Instead of following the traditional approach of defect reporting of writing detailed defect report, the defect will be reported to the developer by simple communication and showing them what the defect is, which they fix quickly and returned to retest.



Summary

- In this lesson, you have learnt
  - The concepts of Agile Testing
  - The different roles and the activities performed by each role in the Agile Team
  - Where does the Tester fit in the Agile Team?
  - We also elaborated the key differences between the traditional testing and agile testing
  - We also elaborated the agile testing process from the User Story perspective
  - How does a change in tester's mind-set facilitates a smooth transition from traditional testing to a agile testing?
  - The changed role of a Test Manager

Summary

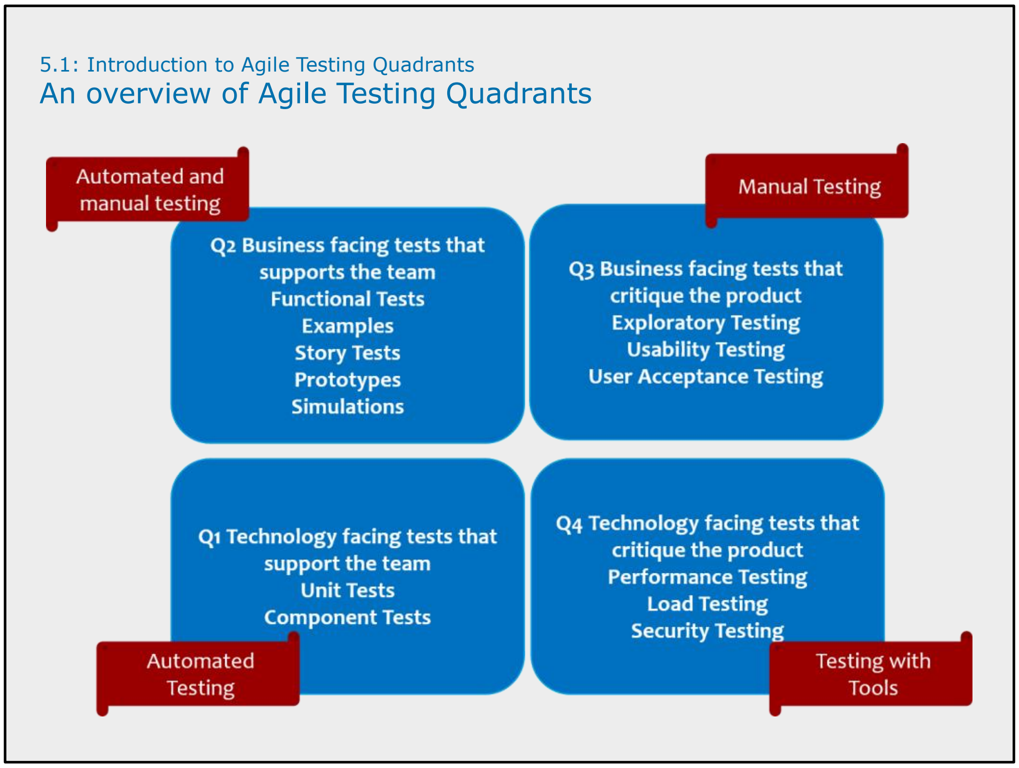
Add the notes here.

Introduction to Agile

Lesson 5: Agile Testing Quadrants and Agile Test Planning

Lesson Objectives

- An overview of Agile Testing Quadrants
- Agile Testing Quadrant 1, 2 & 3 Goals
- Agile Testing Quadrant 1, 2 & 3 Toolkit
- Test Planning in Agile Testing



5.1: Introduction to Agile Testing Quadrants

An overview of Agile Testing Quadrants

Why do we test?

The answer might seem obvious, but in fact, it’s pretty complex. We test for a lot of reasons: to find bugs, to make sure the code is reliable, and sometimes just to see if the code is usable. We do different types of testing to accomplish different goals. Software product quality has many components.

The Agile Testing Quadrants matrix helps testers ensure that they have considered all of the different types of tests that are needed in order to deliver value.

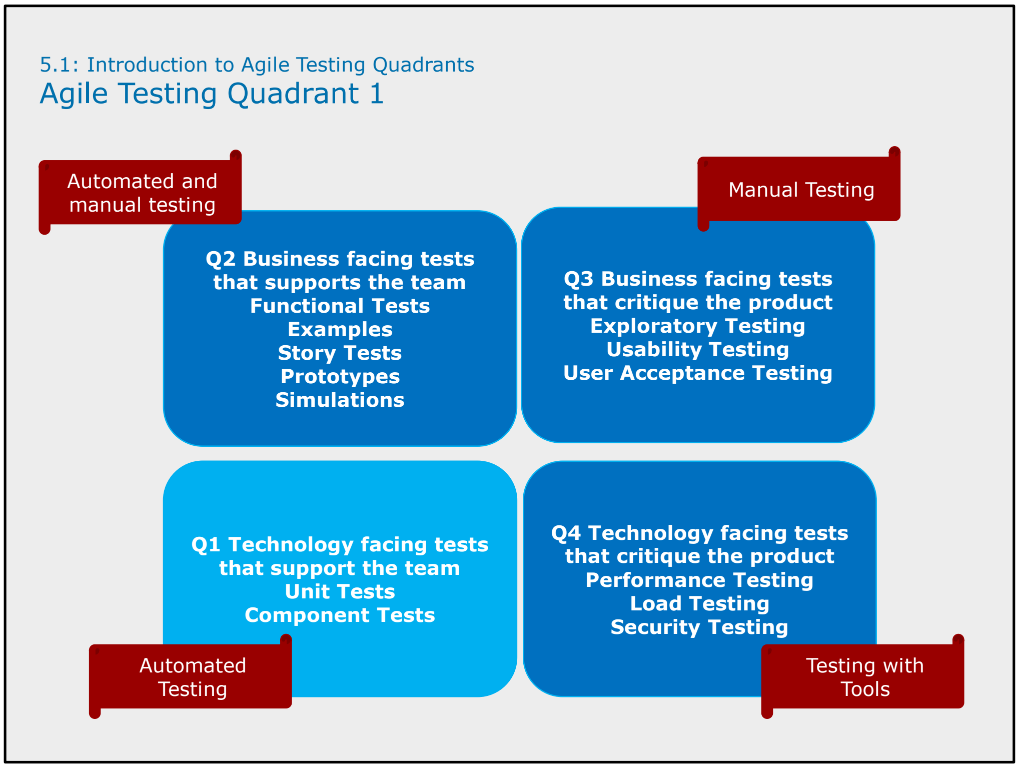
The above diagram of the agile testing quadrants that shows how each of the four quadrants reflects the different reasons we test. On one axis, we divide the matrix into tests that support the team and tests that critique the product. The other axis divides them into business-facing and technology-facing tests.

The order in which the quadrants are numbered is nothing to do with the sequence in which these different types of testing are performed. The time at which various types of tests are performed is dependent on the project risk factor, the product expectations, the type of project i.e. whether the team is working on legacy code or on a greenfield project and most importantly when resources are available to do the testing.

5.1: Introduction to Agile Testing Quadrants

An overview of Agile Testing Quadrants (Cont.)

- Agile development is supported by a bundle of concrete practices, covering areas like requirements, design, modeling, coding, testing, project management, process, quality etc.
- Agile expert Lisa Crispin explains the four Agile testing quadrants and how they can guide managers and development teams in creating a test strategy
- Undoubtedly Agile Testing is gaining its popularity in the testing world with its ability to deliver high quality products that keep the customers satisfied
- Project Managers and teams new to Agile development, the idea of planning and executing all the testing activities within short iterations and release cycles is intimidating
- It is imperative to understand the Agile testing quadrants and how it can help you perform Agile testing better
- Another advantage of the quadrants is that they can explain the whole testing in common language that is easy to understand



5.1: Introduction to Agile Testing Quadrants

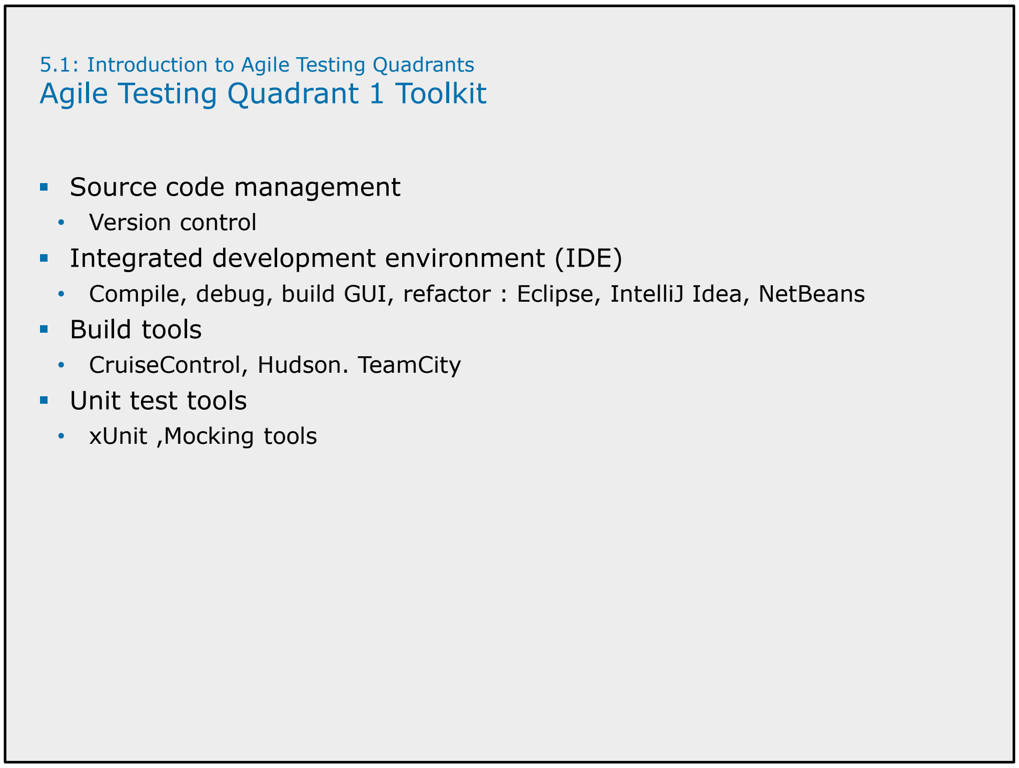
Agile Testing Quadrant 1

Automated and manual testing
Manual Testing
Q2 Business facing tests that supports the team Functional Tests Examples Story Tests Prototypes Simulations
Q3 Business facing tests that critique the product Exploratory Testing Usability Testing User Acceptance Testing
Q1 Technology facing tests that support the team Unit Tests Component Tests
Q4 Technology facing tests that critique the product Performance Testing Load Testing Security Testing
Automated Testing
Testing with Tools

5.1: Introduction to Agile Testing Quadrants

Agile Testing Quadrant 1 Goals

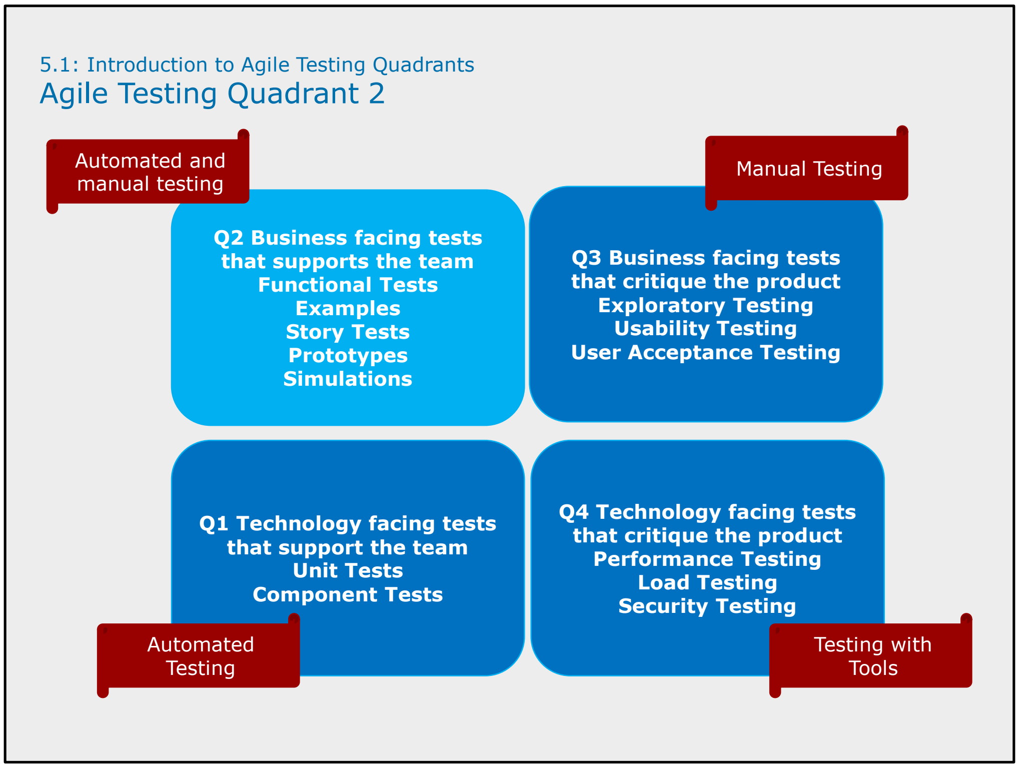
- The lower left quadrant represents test-driven development, which is a core agile development practice
- Unit testing verifies that the functionality of a smallest unit of the system, such as an object or method
- Component testing verifies the behavior of a larger part of the system, such as a group of classes that provide some service
- Both types of tests are usually automated with a member of the xUnit family of test automation tools
- We can refer to these tests as programmer tests, developer facing tests, or technology-facing tests
- They enable the programmers to measure what Kent Beck has called the internal quality of their code



5.1: Introduction to Agile Testing Quadrants

Agile Testing Quadrant 1 Toolkit

- Source code management
  - Version control
- Integrated development environment (IDE)
  - Compile, debug, build GUI, refactor : Eclipse, IntelliJ Idea, NetBeans
- Build tools
  - CruiseControl, Hudson. TeamCity
- Unit test tools
  - xUnit ,Mocking tools



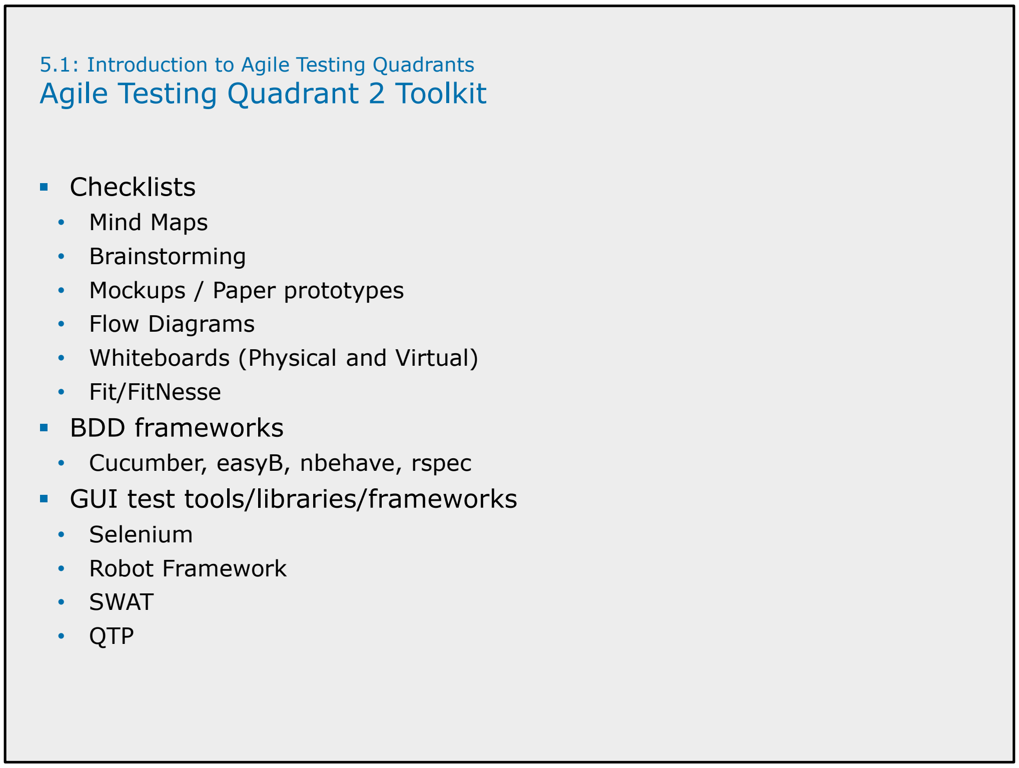
5.1: Introduction to Agile Testing Quadrants

Agile Testing Quadrant 2

5.1: Introduction to Agile Testing Quadrants

Agile Testing Quadrant 2 Goals

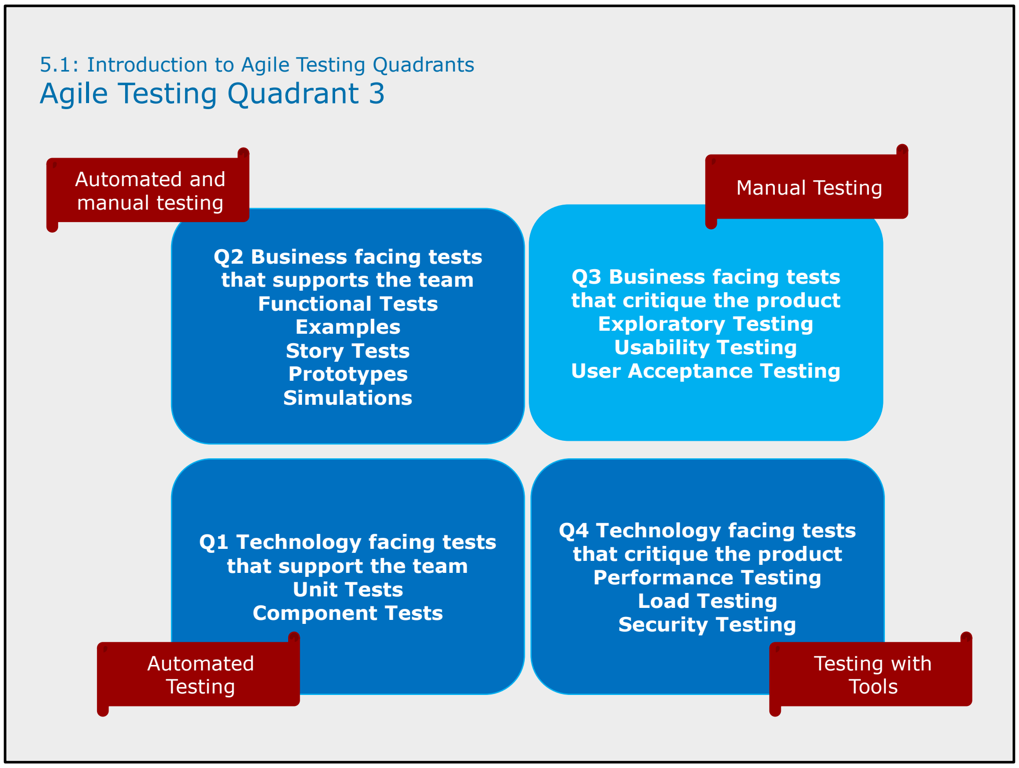
- This quadrant focuses on eliciting the requirements
- The tests in Quadrant 2 support the work of the development team as well, but at a higher level
- These business-facing tests, also called customer-facing tests and customer tests, define external quality and the features that the customers want
- With agile development, these tests are derived from examples provided by the customer team
- They describe the details of each story. Business-facing tests run at a functional level, each one verifying a business satisfaction condition
- The objective of this quadrant is to obtain enough requirements so that coding can commence without any hiccups



5.1: Introduction to Agile Testing Quadrants

Agile Testing Quadrant 2 Toolkit

- Checklists
  - Mind Maps
  - Brainstorming
  - Mockups / Paper prototypes
  - Flow Diagrams
  - Whiteboards (Physical and Virtual)
  - Fit/FitNesse
- BDD frameworks
  - Cucumber, easyB, nbehave, rspec
- GUI test tools/libraries/frameworks
  - Selenium
  - Robot Framework
  - SWAT
  - QTP



5.1: Introduction to Agile Testing Quadrants

Agile Testing Quadrant 3

5.1: Introduction to Agile Testing Quadrants

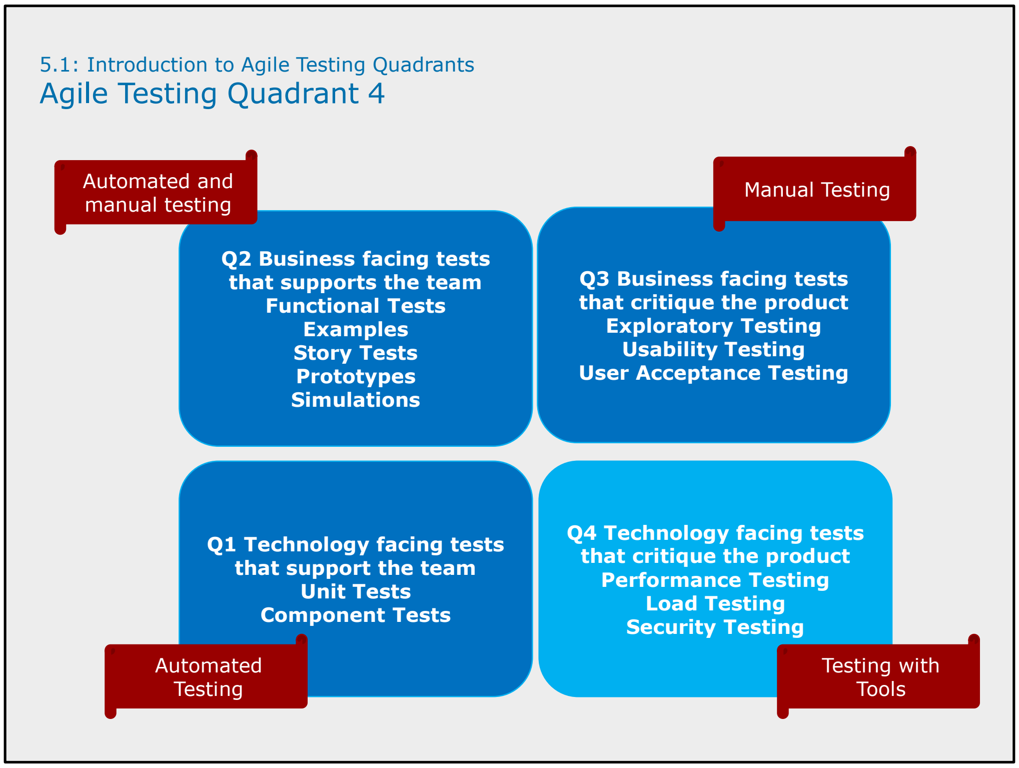
Agile Testing Quadrant 3 Goals

- Quadrant 3 classifies the business-facing tests that exercise the working software to see if it doesn't quite meet expectations or won't stand up to the competition
- When we do business-facing tests to critique the product, we try to emulate the way a real user would work on the application
- This is manual testing that only a human can do
- User Acceptance Test (UAT) gives customers confidence when they see requirements are met
- The benefits of UAT significantly outweigh the investments
- Exploratory testing is central to this quadrant
- During exploratory testing sessions, the tester simultaneously designs and performs tests, using critical thinking to analyze the results
- This offers a much better opportunity to learn about the application than scripted tests

5.1: Introduction to Agile Testing Quadrants

Agile Testing Quadrant 3 Toolkit

- Generate test data
  - e.g. PerlClip, Ruby script
- Emulators
  - Duplicate system behavior
  - e.g. mobile devices



5.1: Introduction to Agile Testing Quadrants

Agile Testing Quadrant 4

5.1: Introduction to Agile Testing Quadrants

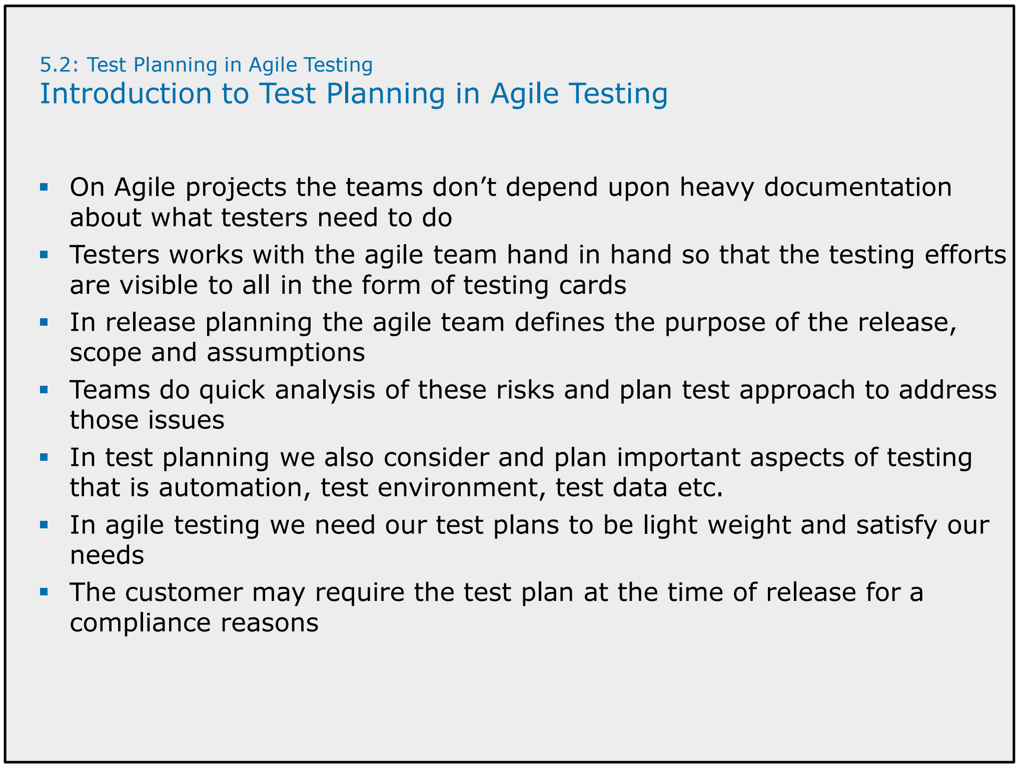
Agile Testing Quadrant 4 Goals

- Technology-facing tests in Quadrant 4 are intended to critique product characteristics such as performance, robustness, and security
- This quadrant is responsible to deliver the ultimately finished product
- The application is made to deliver the expected value and non-functional qualities with help of this quadrant
- Kind of tests in this quadrant
  - Non-functional tests such as stress and performance testing
  - Security testing with respect to hacking and authentication
  - Infrastructure testing
  - Data migration testing
  - Scalability testing
  - Load testing
  - Maintainability Test
  - Compatibility Test
  - Data Migration Testing
  - Recovery Testing

5.1: Introduction to Agile Testing Quadrants

Agile Testing Quadrant 4 Toolkit

- Monitoring tools examples
  - jConsole
- Application bottlenecks, memory leaks
  - jProfiler
  - Database usage
- Commercial load test tools
  - Loadrunner
  - Silk Performer
- Open source test tools
  - jMeter
  - The Grinder
  - jUnitPerf
- Performance test providers
  - Multiple sites



5.2: Test Planning in Agile Testing

Introduction to Test Planning in Agile Testing

- On Agile projects the teams don’t depend upon heavy documentation about what testers need to do
- Testers works with the agile team hand in hand so that the testing efforts are visible to all in the form of testing cards
- In release planning the agile team defines the purpose of the release, scope and assumptions
- Teams do quick analysis of these risks and plan test approach to address those issues
- In test planning we also consider and plan important aspects of testing that is automation, test environment, test data etc.
- In agile testing we need our test plans to be light weight and satisfy our needs
- The customer may require the test plan at the time of release for a compliance reasons

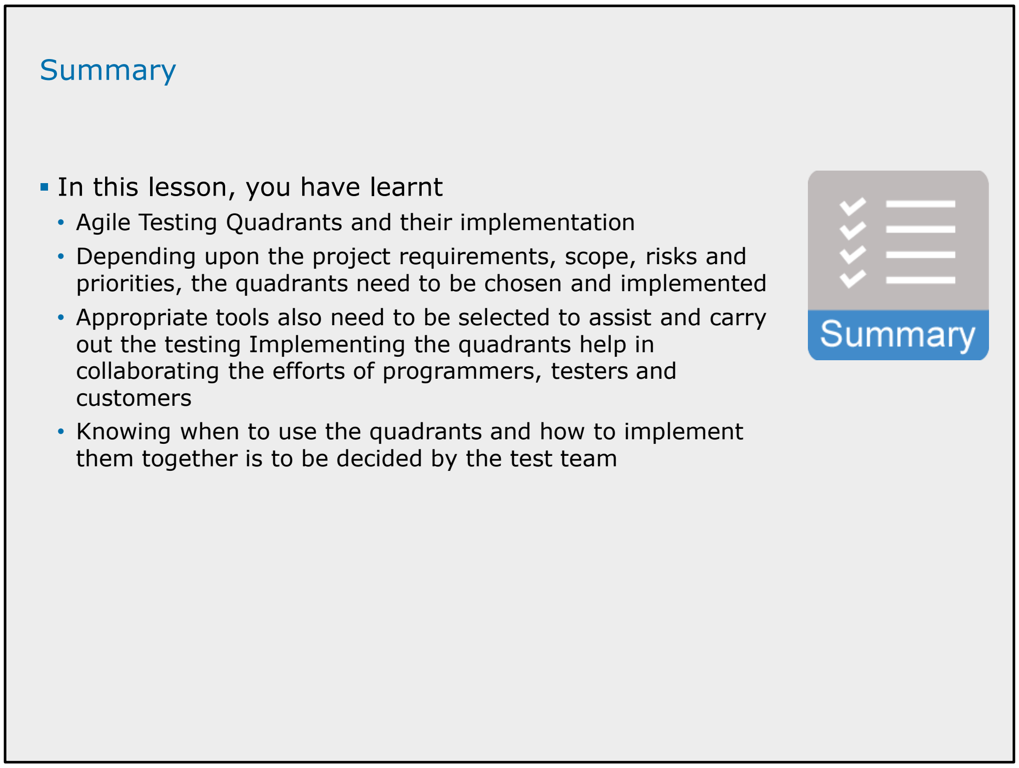
Test Planning in Agile Testing

In traditional software development methodologies a good amount of importance is given to the test plan documentation. The test plans are intended to outline the objective, scope, approach and focus of the software testing efforts for stakeholders. The completed document help people outside the testing group understand the “WHY” and “HOW” of the product validation.

5.2: Test Planning in Agile Testing

Introduction to Test Planning in Agile Testing (Cont.)

- The test plan should contain information about testing issues that are specific to this release or project
- It should contain risk analysis and assumptions identified
- The test plan should outline the critical success factors that your customers has identified
- Keep only that information which people need to know about testing and remove any extraneous information
- The main advantage of test planning is planning itself
- It allows you to consider and address issues such as test data requirements, infrastructure
- Test planning is a risk mitigation strategy



Summary

- In this lesson, you have learnt
  - Agile Testing Quadrants and their implementation
  - Depending upon the project requirements, scope, risks and priorities, the quadrants need to be chosen and implemented
  - Appropriate tools also need to be selected to assist and carry out the testing Implementing the quadrants help in collaborating the efforts of programmers, testers and customers
  - Knowing when to use the quadrants and how to implement them together is to be decided by the test team

Summary

Add the notes here.