

MODULE 1

Agile Practices

Facilitator Notes:

Welcome the participants and give them an overview of the module. Tell them that they will learn about the 'Agile Practices' in this module.

You will learn about the 'Agile Practices' in this module.

Module Objectives

At the end of this module, you will be able to:

- Learn Agile methodology
- Define software, history of software engineering, and software development methodologies
- Identify the traditional software development models
- Discuss the waterfall model and classical waterfall model
- Understand about traditional IT organizations
- Learn about developers vs IT operations conflict
- Explain birth of Agile, and four values of the Agile manifesto



- Describe about Scrum, Scrum theory, Scrum values, Scrum roles, Scrum master, Scrum sprints, benefits of Scrum, etc.
- Describe planning and estimation, Agile planning, and its levels
- Define conditions of satisfaction and velocity
- List the various estimating techniques
- Discuss about soft skills in Agile
- Explain the Kanban model

Module Topics

Let us take a quick look at the topics that we will cover in this module:

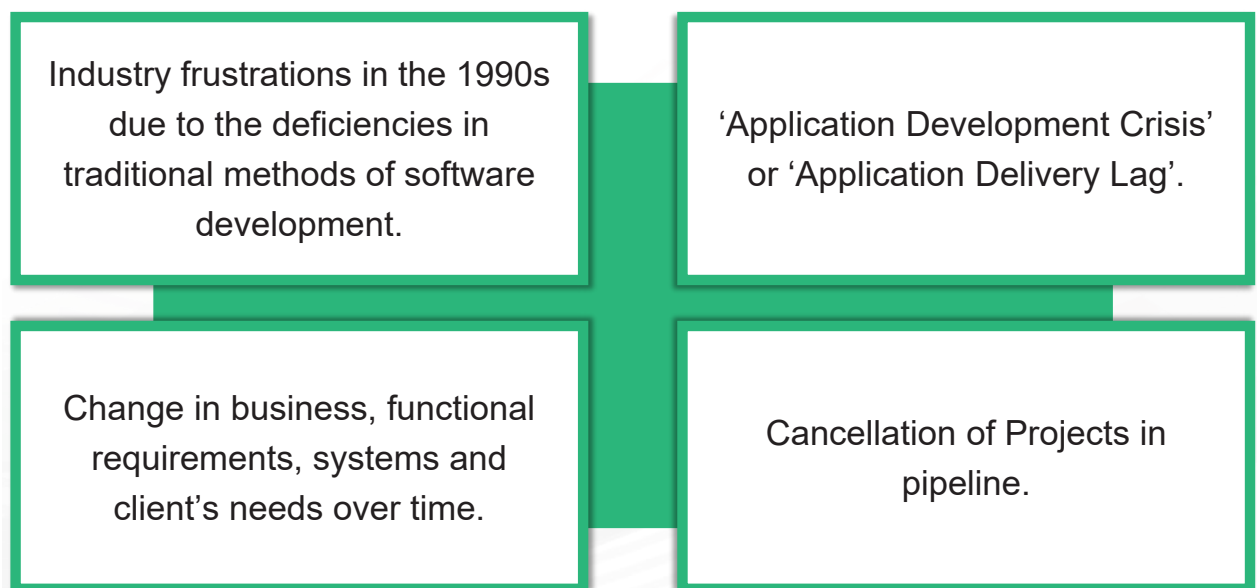
- Agile methodology
- Software, History of Software Engineering and Software Development Methodologies
- Traditional Software Development Models
- Waterfall Model, Classical Waterfall Model
- Traditional IT Organizations
- Developers vs IT Operations Conflict
- Birth of Agile, Four Values of the Agile Manifesto
- Scrum, Scrum Theory, Scrum Values, Scrum Roles, Scrum Master Scrum Sprints, Benefits of Scrum
- Planning and Estimation, Agile Planning, Levels of Agile Planning
- Conditions of Satisfaction, Velocity



- Estimating Techniques
- Soft Skills in Agile
- Kanban Model

1.1 Agile Methodology - Evolution

The following diagram illustrates the evolution of Agile methodology.



Facilitator Notes:

- Ask the participants what they think, triggered the rise of Agile.
- Note down, what they say on the slide.
- Then show the boxes with the reasons.

Around the 1990s, the software development industry was in a great crisis. Due to the deficiencies in the traditional software development approaches, there was an enormous lag in time between the functional requirements requested by the customers and the delivery of technology. At that time, it was estimated that the time period between business need validation and the delivery of software/application was around three years. Some businesses faced a lag of more than three years.

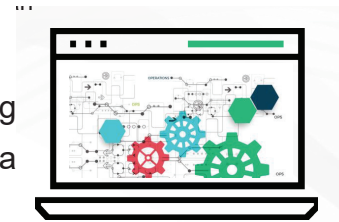
Three years is a more than sufficient time period for businesses, systems and requirements to change. This resulted in the cancellation of projects mid-way. Completed projects failed to make sense, because of the change in requirements over time.

Industry experts were forced to develop alternative methods of software development and delivery, and this laid the foundation for the birth of Agile.

1.2 Software

Following are the key details of the software:

- A software is an organized information in the form of operating systems, programs, utilities and applications that enable a computer to work.
- Software comprises a set of machine-readable instructions that directs a computer's processor to perform specific operations.
- It includes computer programs, libraries and their associated documentation.
- Software programs are stored as files on a storage device such as hard disk, DVD or memory sticks. When needed, they are loaded into the computer's memory (RAM).
- In a nutshell, a software is everything that governs the functioning of a computer, an interface between computers and humans who use them.



Facilitator Notes:

Explain participants what a software is and why it is required.

Computers and mobile phones have become an inevitable part of everyone's life. On a daily basis, all of us interact with operating systems, spreadsheets, documents, games, videos and so many other applications. Ever wondered how all these things function? Behind each and every application that we use, lies a software built by developers, on multiple different programming languages. Let's look at what a software is, how it works, different types of software, the history of software engineering and the different methods of software development.

What is a Software?

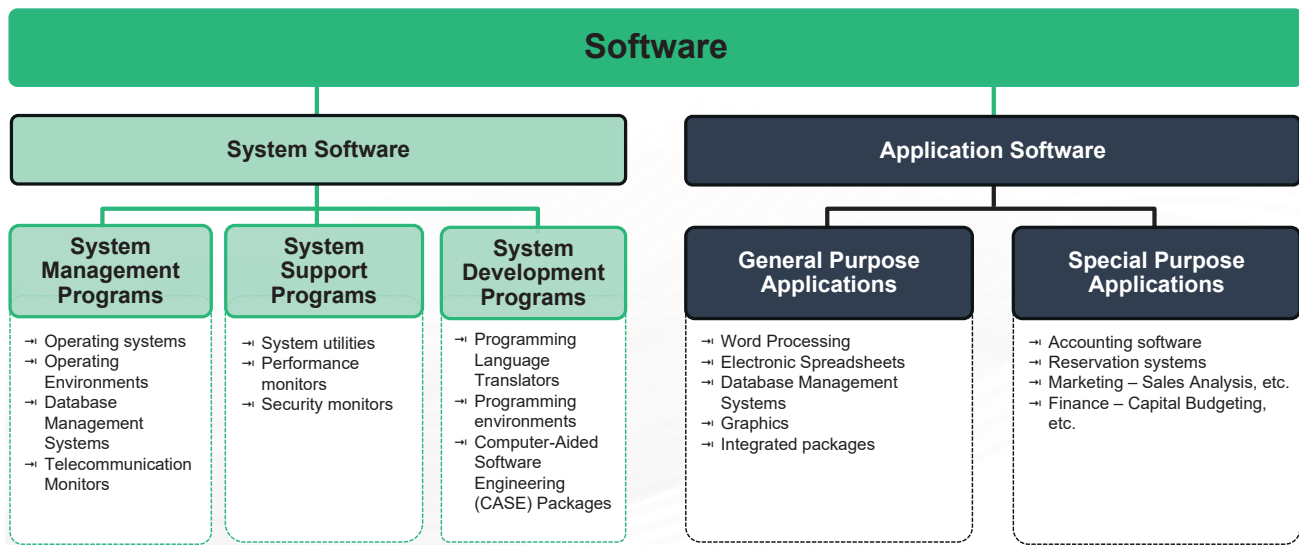
A software can be defined as an organized information in the form of operating systems, programs, utilities and applications that enable a computer to work. A software consists of carefully-organized instructions and code written by programmers in any of the different programming languages. A software is different from the physical hardware (from which the computer system is built), in a way that it contains the data or instructions which enable the system to perform. A software includes computer programs, libraries and other related non-executable data, such as online documentation or digital media.

Software controls the computer (operating systems), the communication of information (networks), and the creation and control of other programs (software tools and environments). The software also transforms information in different ways as producing, managing, acquiring, modifying, displaying, or transmitting information, whether it is a computer desktop or a mobile phone.

A computer requires both hardware and software for its function. Without either, the other might not work on its own. For example, without a physical device like desktop computer or mobile phone you will not be able to use the Internet, or an operating system, the browser could not run on the computer.

In today's context, software takes on a dual role, it is a product and at the same time, the vehicle for delivering the product. As a product, it delivers the computing potential embodied by computer hardware or more broadly, by a network of computers that are accessible by local hardware. Whether it resides within a mobile phone or operates inside a mainframe computer, the software is an information transformer—producing, managing, acquiring, modifying, displaying, or transmitting information that can be as simple as a single bit or as complex as a multimedia presentation derived from data acquired from dozens of independent sources. As the vehicle used to deliver the product, software acts as the basis for the control of the computer (operating systems), the communication of information (networks), and the creation and control of other programs (software tools and environments).

1.2.1 Different Types of Software



Facilitator Notes:

Explain participants about the different types of software.

We've now learnt that a software is needed for functioning of computer systems. We will now have a look at the different types of software.

There are two major categories of software, based on the purpose of use:

1. System Software
2. Application Software

System Software

System software is the one that operates the computer hardware and enables the functioning of the computer system. It includes operating systems, device drivers, diagnostic tools, servers, utilities, etc. The major purpose of system software is to insulate the application programmer to the extent possible from the details of the particular computer complex being used, especially memory and other hardware features, such as accessory devices as communications, printers, readers, displays, keyboards, etc. System software provides the platform for running application software. Systems software area is characterized by heavy

interaction with computer hardware; heavy usage by multiple users; concurrent operation that requires scheduling, resource sharing, and sophisticated process management; complex data structures; and multiple external interfaces.

System software in turn can be classified into:

a) System Management Programs

Programs that are responsible for the functioning and management of computer systems. These are the ones that run and manage the resources and provide common services for other software that run on top of them. System management programs include:

- Operating systems
- Operating Environments
- Database Management Systems
- Telecommunication Monitors

b) System Support Programs

It is a program that supports, or facilitates the smooth and efficient execution of various programs and operations of a computer. System support programs include:

- System utilities
- Performance monitors
- Security monitors

c) System Development Programs

System development programs have the instructions to create and maintain computer systems. These include:

- Programming Language Translators
- Programming environments
- Computer-Aided Software Engineering (CASE) Packages


Application software

A group of programs designed to carry out a single or a group of related tasks. Application software is actually a subclass of computer software, which leverages the capabilities of a computer directly to a task that the user wishes it to perform. Application software is looked upon as a software as well as its implementation. Application software is stand-alone programs that solve a specific business need. Applications in this area process business or technical data in a way that facilitates business operations or management/technical decision making. In addition to conventional data processing applications, application software is used to control business functions in real time.

Application software can be broadly classified into:

- a) **General purpose software:** General purpose application software is a type of application that can be used for a variety of tasks. It is not limited to one particular function. Different types of general purpose software include:
 - Word processing software
 - Electronic spreadsheets
 - Database managers
 - Presentation graphics
 - Integrated packages
- b) **Special purpose software:** Special purpose application software is a type of software created to execute one specific task. Some examples of special purpose software include:
 - Accounting software
 - Reservation systems
 - Marketing – Sales Analysis, etc.
 - Finance – Capital Budgeting, etc.

What did You Grasp?



1. Which of the following is not a system software?

- A) Operating software
- B) Performance monitor
- C) Database management systems
- D) Word processing software

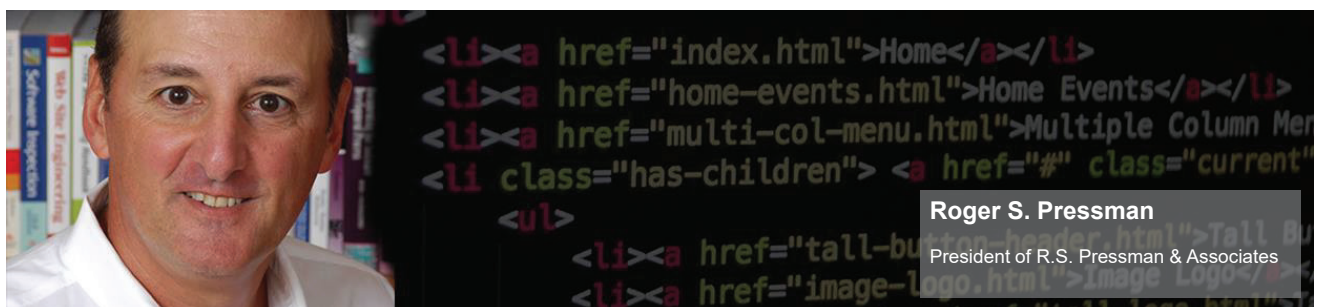
Facilitator Notes:

Tell the participants that they will be going through a knowledge check question.

Answer:

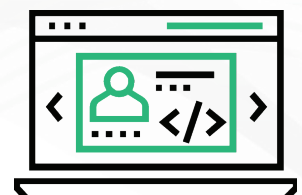
1. d. Word processing software

1.3 Definition of Digital Transformation



"Software engineering is the technology that encompasses a process, a set of methods and an array of tools that allow professionals to build high quality computer software."

Source adopted from "Software Engineering: A Practitioner's Approach" written by Roger S. Pressman



Facilitator Notes:

Give a brief introduction of software engineering and provide some definitions.

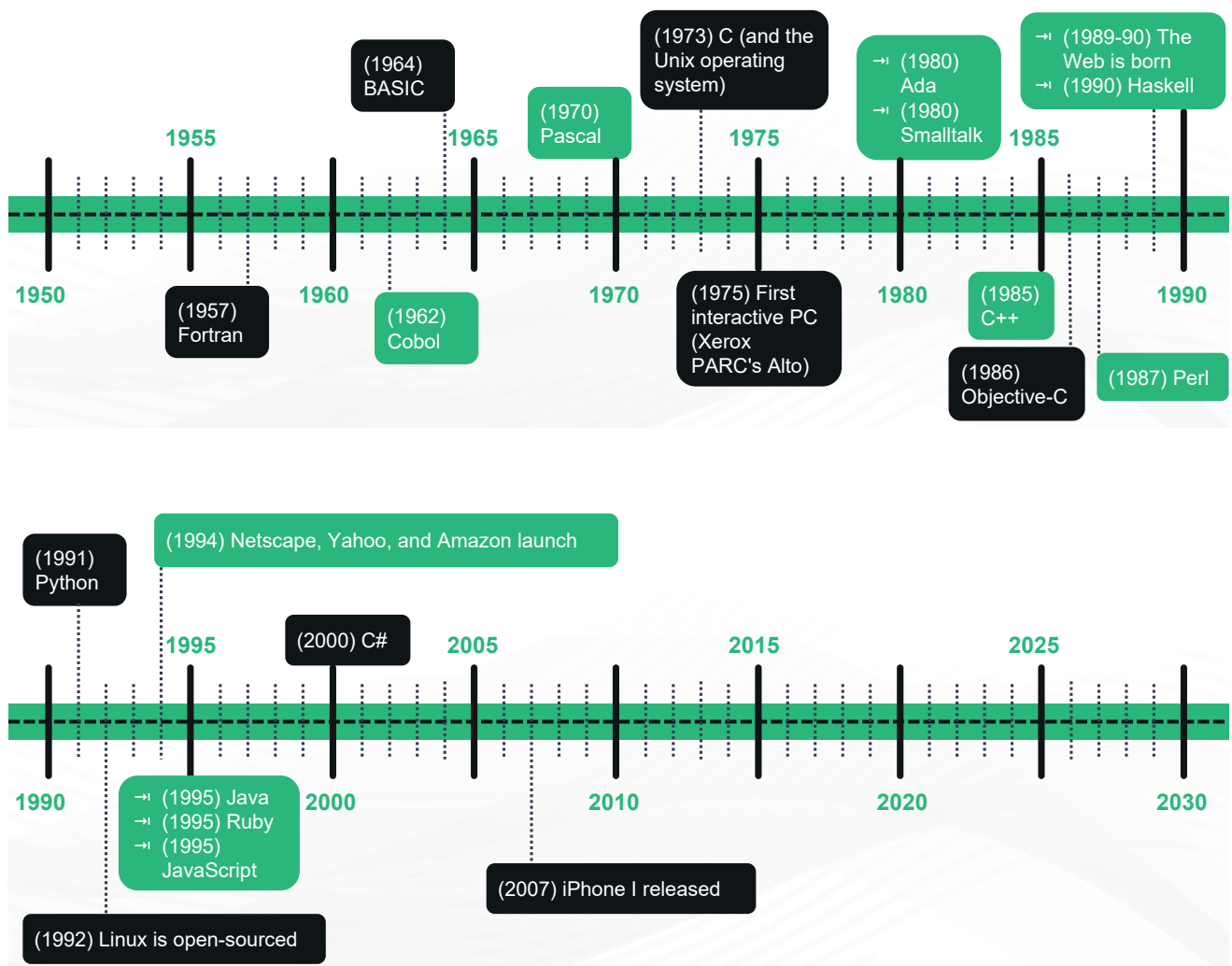
Software engineering is a detailed study of engineering to the design, development and maintenance of software. Software is important because it affects nearly every aspect of our lives and has become pervasive in our commerce, culture and everyday activities. Software engineering is important because it enables us to build complex systems in a timely manner and with high quality.

Definition of Software Engineering

- IEEE Standard Glossary of Software Engineering Terminology: “The application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software.”
- IEEE Systems and software engineering - Vocabulary: “The systematic application of scientific and technological knowledge, methods, and experience to the design, implementation, testing, and documentation of software.”
- Definition by Roger S. Pressman (American software engineer, author and consultant, and President of R.S. Pressman & Associates): “Software engineering is the technology that encompasses a process, a set of methods and an array of tools that allow professionals to build high quality computer software.”
- Definition by Ian Sommerville (British academic and author of text books): “An engineering discipline that is concerned with all aspects of software production.”

As per these definitions, a software may be very complex and beyond the handling of a single individual. As a result, a number of people are expected to work on the pieces of a software product in a cooperative manner. Also, a software product can have many versions which need to be managed. We can say, Software engineering is also a management activity in addition to the implementation activity.

1.4 History of Software Engineering



Facilitator Notes:

Explain to participants the classification of software engineering in various areas.

The history of software engineering is rich and fascinating. Software engineering began when computer programs were just instructions to manipulate a physical device and it has crossed several key turning points that led to the commercialization and the consumerization of computing technology.

Initial Days of Software

It was the computer scientist Tom Kilburn, who wrote the world's first piece of software, run in 1948 at the University of Manchester in England. Kilburn and his colleague Freddie Williams had built one of the earliest computers, the Manchester Small-Scale Experimental Machine (also known as the "Baby"). The SSEM was programmed to perform mathematical calculations using machine code instructions. This first piece of software took 52 minutes to correctly compute the greatest divisor of 2 to the power of 18 (262,144).

Back in the late '50's and early '60's, the programmers didn't interact directly with computing devices. They delivered their programs by hand to technicians and picked up the results hours later, after the programs were batch processed with many others. Thus, early tasks were typically geared towards mathematical computation, which required a very limited feedback loop.

For decades after this groundbreaking event, computers were programmed with punch cards in which holes denoted specific machine code instructions. Fortran, one of the very first higher-level programming languages, was originally published in 1957 by IBM, for mathematical and scientific computing. The next year, statistician John Tukey coined the word 'software' in an article about computer programming. Another programming language Cobol, was released by the US Department of Defense in 1962 for use in business applications. Other pioneering programming languages like BASIC, Pascal and C arrived over the next two decades.

The Software Crisis

The transition to using a time-sharing model instead of batch processing for running programs was perhaps most significant of all because it led to a rapid growth in computing applications. Unfortunately, projects consistently failed to deliver reliably, on time and on budget. Practitioners were forced to admit that they lacked the proper best practices to implement and produce software at scale commercially. They called it the 'Software Crisis'.

Foundations of Software Engineering

The software crisis period taught a great lesson that designing complex software systems would require better tools and approaches than were available at the time. A conference was convened in 1968 to find a solution. It was at this conference, where the term 'Software Engineering' found its roots. The conference sought to apply the best practices of project

management and production to software. As a result, they produced a report which defined the foundations of software engineering. The early 70's saw the emergence of key ideas in systems thinking which allowed engineers to break these giant projects into modular (and much more manageable) pieces that communicated via interfaces.

The Personal Computing Era

In the 1970s and 1980s, software became a boom with the arrival of personal computers. C, the general-purpose programming language was originally developed by Dennis Ritchie between 1969 and 1973 at Bell Labs, and used to re-implement the Unix operating system. Apple released Apple II, its revolutionary product, to the public in April 1977. VisiCalc, the first spreadsheet software for personal computing, was wildly popular and known as the Apple II's killer app. The software was written in a specialized assembly language and appeared in 1979. The IBM PC was first launched in 1981. The next year, Time magazine selected the personal computer as its Man of the Year.

Software for productivity and business dominated these early stages of personal computing as well. Many significant software applications, including AutoCAD, Microsoft Word and Microsoft Excel, were released in the mid-1980s. Between 1980 and 1995, the major programming languages and frameworks like C++, Objective C, Perl, Haskell, Python and Java were released, and this period brought a major breakthrough in the industry.

With the advent of the Internet, open-source software, another major innovation in the history of software development, first entered the mainstream in the 1990s. The Linux kernel, which became the basis for the open-source Linux operating system, was released in 1991. Interest in open-source software spiked in the late 1990s, after the 1998 publication of the source code for the Netscape Navigator browser, mainly written in C and C++. Also, noteworthy is the release of Java by Sun Microsystems in 1995.

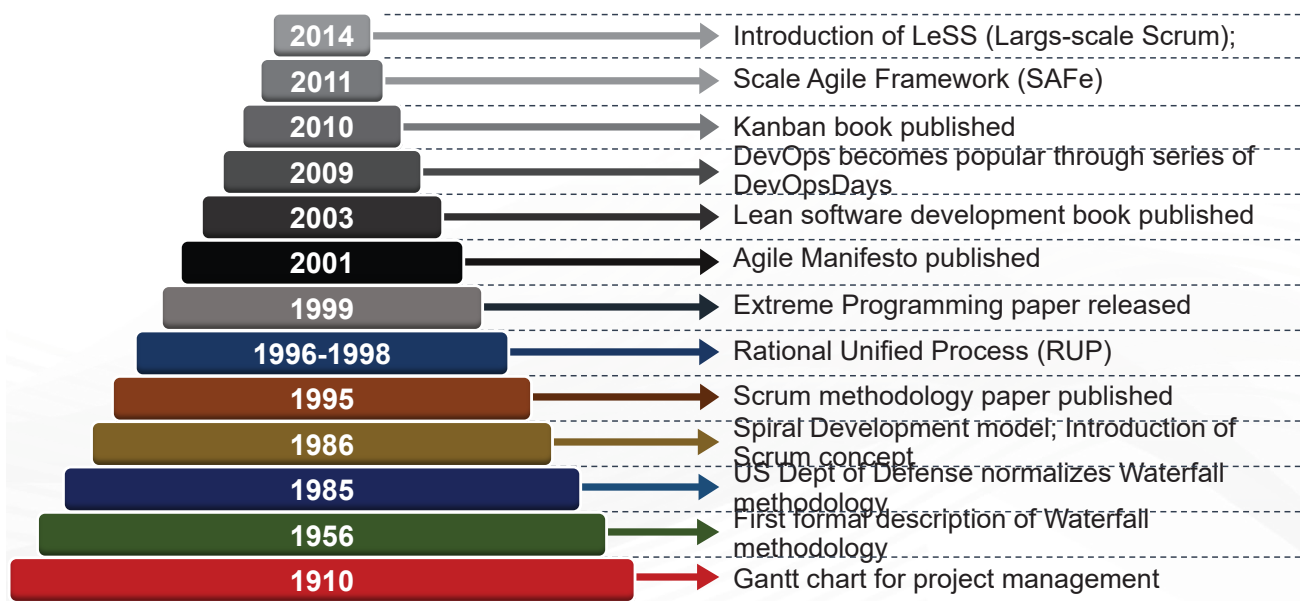
Mobile Computing

In 1993, IBM first released the smartphone and in 1996, Palm OS became a hit in the market. In 1999, RIM released the very first Blackberry 850 device. In 2007, Apple changed computing with the release of the iPhone. It was only after this period, Mobile computing and mobile applications began to explode. Mobile apps use Swift (iOS) and Java (Android) as the development languages.

Some programming languages, like C and Cobol, have survived the test of time and are still in use. Other languages, such as Java and Python, are somewhat younger and have been used in countless software development projects. Still others, such as Apple's Swift programming language for iOS or Go Open source, are relatively new and exciting.

Let's look at some of the traditional ways of software development.

1.5 History of Software Development Methodologies



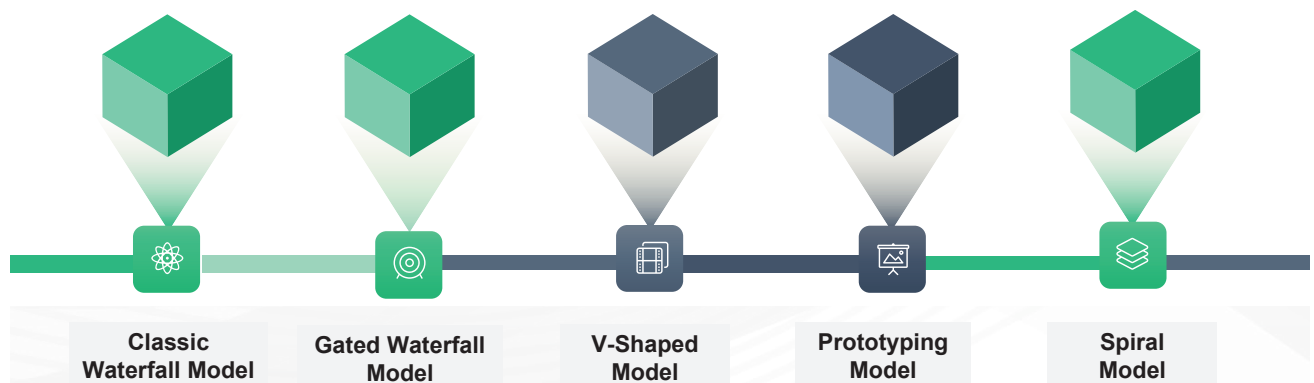
Facilitator Notes:

Explain briefly about the sequence of events in the history of software development.

Go through the above infographic to understand the major milestones in the history of software development methodologies.

1.6 Traditional Software Development Models

Many life cycle models have been proposed so far, with their own advantages and disadvantages. A few important traditional software development models are as follows:



Facilitator Notes:

Explain the participants that 'Software development life cycle (SDLC)' consists a series of phases that provides a common understanding of the software building process. Let the participants know that they will also learn about - How software is realized and developed from the business understanding and requirements elicitation phase to convert these business ideas and requirements into functions and features until its usage and operation to achieve the business needs.

Traditionally, the software was designed by organizing the development process into a series of phases, that will happen one after the other. The output of one phase will be the input for the next phase. In most of the cases, it is unidirectional, starting from design requirement to development to deployment, with no reversal in the flow. This meant that any change in the requirement caused the process to start all over again, and any bug or error caused serious issues in terms of time, effort and cost.

The following list gives an overview of a few traditional software development models. Each model comes with its own set of advantages or disadvantages. A few important and commonly used life cycle models are as follows:

- **Classical Waterfall Model:** The waterfall model emphasizes that a logical progression of steps be taken throughout the software development life cycle (SDLC), much like the cascading steps down an incremental waterfall. While the popularity of the waterfall model has waned over recent years in favor of more agile methodologies, the logical nature of the sequential process used in the waterfall method cannot be denied, and it remains a common design process in the industry. The steps involved in the classical waterfall model are as follows:
 - Requirements
 - Analysis

- Design
- Coding
- Testing
- Operations/Maintenance
- **Gated Waterfall Model:** In a gated waterfall model, there is a quality gate between the phases. The quality gate is based on the review and acceptance of artifacts. The feedback can also go back to any phase. The gated waterfall model is explained in a later section.
- **V-shaped Model:** An extension to the waterfall model. The process flow doesn't move down in a linear way. Instead, the process flow bends and moves upwards after implementation and coding phase, to form the typical V-shape. The major difference between waterfall model and V-model is that test planning and designing happen in early stages well before coding. This makes the model more successful than the classic waterfall model, as it saves a lot of time. This model suits well for small projects, where requirements are clearly defined and easily understood. The steps involved in this model are as follows:
 - Requirements
 - System design
 - Architecture design
 - Module design
 - Implementation/Coding
 - Unit testing
 - Integration testing
 - System testing
 - Acceptance testing
- **Prototyping Model:** The Prototyping Model is one of the most popularly used Software Development Life Cycle Models (SDLC models). This model is used when the customers do not know the exact project requirements beforehand. In this model, a prototype of the end product is first developed, tested and refined as per customer feedback repeatedly till a final acceptable prototype is achieved which forms the basis for developing the final product. There are two approaches:
 - **Rapid throwaway prototyping:** Prototypes are first created, but they will not become a part of the finally delivered software.
 - **Evolutionary prototyping:** Prototypes gradually evolve as the final product by means of an iterative incorporation of user feedback.

- **Incremental prototyping:** The final product is built as individual prototypes. In the end, the prototypes are merged in an overall design to get the final product.
- **Extreme prototyping:** Most commonly used for developing web applications. It involves three phases:
 - Phase 1: A static prototype is built that mainly contains HTML pages
 - Phase 2: Screens are programmatically converted to a fully functional product using a simulated services layer.
 - Phase 3: Services are implemented during this phase.
- **Spiral Model:** The Spiral Model is a software development methodology that aids in choosing the optimal process model for a given project. It combines aspects of the incremental build model, waterfall model and prototyping model. The Spiral Model is concerned primarily with risk awareness and management. Spiral model is distinguished by a set of six invariant characteristics:
 - Define artifacts concurrently
 - There are four essential spiral tasks
 - Risk determines level of effort
 - Risk determines degree of details
 - Use the anchor point milestones
 - Focus on the system and its life cycle

What did You Grasp?



1. In which of the following prototyping methods, initially created prototypes will not become part of the final version of the software?
 - A) Extreme prototyping
 - B) Rapid throwaway prototyping
 - C) Evolutionary prototyping
 - D) Incremental prototyping
2. Which of the following statements about Spiral model is true?
 - A) Define artifacts sequentially
 - B) There are four essential spiral tasks
 - C) Risk determines level of effort
 - D) Risk determines degree of details

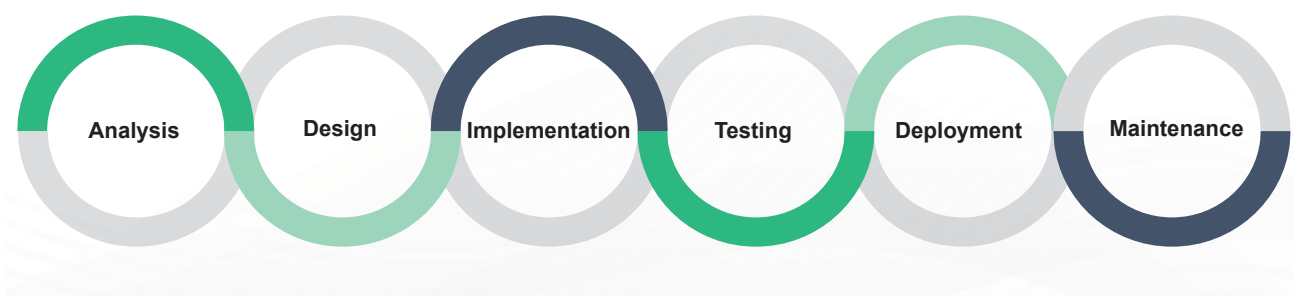
Facilitator Notes:

Tell the participants that they will be going through a knowledge check question.

Answers:

1. b. Rapid throwaway prototyping
2. a. Define artifacts sequentially

1.7 Waterfall Model



Facilitator Notes:

Let the participants know that in the classical waterfall model the development process begins only if the previous phase is complete. Explain to the participants why this model has been so popular.

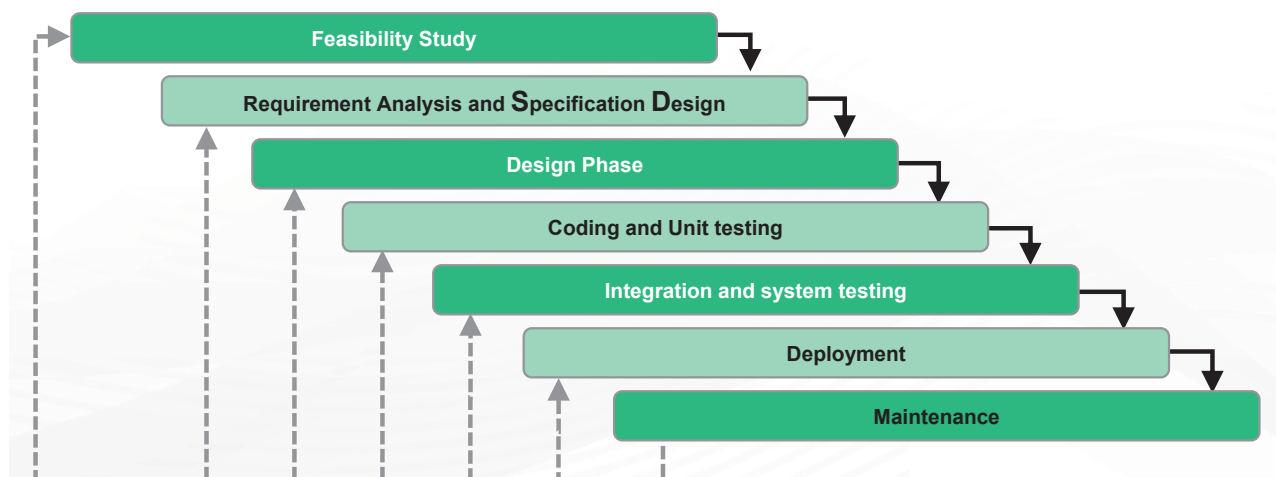
The most popular way of developing a software is the 'THE CLASSICAL WATERFALL' model . It is the first SDLC model, introduced to describe the software development, in late 1950 and became popular in 1970's by the first formal description of the waterfall model cited in an article by Winston W. Royce though the term 'waterfall' was coined by Bell and Thayer in 1976.

In waterfall model, the entire software development process undergoes different phases, namely requirement details/analysis, design, coding and testing, system testing and integration, deployment and maintenance.

A team of experts and trained people in each department handle different phases of the Waterfall model. For example, business and requirements analysis department, software engineering department, development and programming department, quality assurance (QA) department, and technical support department.

1.8 Classical Waterfall Model

In the waterfall approach, the outcome of one phase acts as the input for the next phase sequentially. The following picture illustrates the flow of steps in the waterfall model.



Facilitator Notes:

Let the participants know that in classical waterfall model the development process begins only if the previous phase is complete. Explain to the participants why this model has been so popular.

In the waterfall approach, the whole process of software development is divided into separate phases. The outcome of one phase acts as the input for the next phase sequentially. This means that any phase in the development process begins only if the previous phase is complete. The waterfall model is a sequential design process in which progress is seen as flowing steadily downwards (like a waterfall) through the phases of conception, initiation, analysis, design, construction, testing, production/implementation and maintenance.

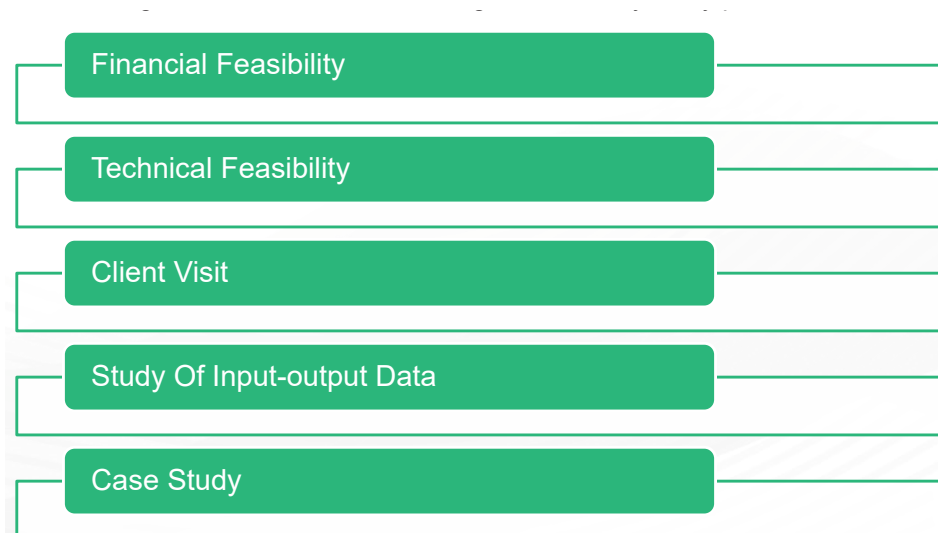
As the Waterfall Model illustrates the software development process in a linear sequential flow; hence it is also referred to as a Linear-Sequential Life Cycle Model.

The sequence of steps in the waterfall model is as follows:

1. The first and foremost stage is the study of resources, technical & financial feasibility.
2. Requirement Analysis and specification design
3. Design Phase
4. Coding and Unit testing
5. Integration and system testing
6. Maintenance

1.8.1 First Stage – Feasibility Study

The following activities are covered during the feasibility study phase:



Facilitator Notes:

Explain the participants the objective of feasibility study and how it is helpful to determine the durability of the project.

The main aim of feasibility study is to determine whether it would be financially and technically feasible to develop the product. At first project managers or team leaders figure out a rough understanding of what is required to be done by visiting the client. They study different input and output data to be produced by the system. They study the required processing to be done on these data and look at the various constraints and effects on the behavior of the system. After understanding the problem (if any) they investigate the different available solutions.

Then they examine each of the solutions in terms of what kind of resources required, what would be the cost of development and what would be the development time for each solution.

Based on this analysis they pick the best solution and determine whether the solution is feasible financially and technically. They check whether the customer budget would meet the cost of the product and whether they have sufficient technical expertise in the area of development. The following is an example of a feasibility study undertaken by an organization. It is intended to give you a feel of the activities and issues involved in the feasibility study phase of a typical software project.

Case Study:

A mining company named Galaxy Mining Company Ltd. (GMC) has mines located at various places in India. It has about fifty different mine sites spread across eight states. The company employs a large number of mines at each mine site. Mining being a risky profession, the company intends to operate a special provident fund, which would exist in addition to the standard provident fund that the miners already enjoy. The main objective of having the special provident fund (SPF) would be quickly distributed some compensation before the standard provident amount is paid. According to this scheme, each mine site would deduct SPF installments from each miner every month and deposit the same with the CSPFC (Central Special Provident Fund Commissioner). The CSPFC will maintain all details regarding the SPF installments collected from the miners. GMC employed a reputed software vendor Adventure Software Inc. to undertake the task of developing the software for automating the maintenance of SPF records of all employees. GMC realized that besides saving manpower on bookkeeping work, the software would help in speedy settlement of claim cases. GMC indicated that the amount it can afford for this software to be developed and installed is Rs. 1 million.

Adventure Software Inc. deputed their project manager to carry out the feasibility study. The project manager discussed the matter with the top managers of GMC to get an overview of the project. He also discussed the issues involved with the several field PF officers at various mine sites to determine the exact details of the project. The project manager identified two broad approaches to solve the problem. One was to have a central database which could be accessed and updated via a satellite connection to various mine sites. The other approach was to have local databases at each mine site and to update the central database periodically through a dial-up connection. These periodic updates could be done on a daily or hourly basis depending on the delay acceptable to the GMC in invoking various functions of the software. The project manager found that the second approach was very affordable and more fault-tolerant as the local mine sites could still operate even when the communication link to the central database temporarily failed. The project manager quickly analyzed the database functionalities required, the user-interface issues, and the software handling communication with the mine sites. He arrived at a cost to develop from the analysis. He found that the solution involving maintenance of local databases at the mine sites and periodic updating of a central database was financially and technically feasible. The project manager discussed his solution with the GMC management and found that the solution was acceptable to them as well.

1.8.2 Second Stage – Requirements Analysis and Specification Design

The two major activities carried out during the requirements analysis and specification design phase.

Requirements gathering and analysis

- All relevant information regarding the product is collected from the customer and users through interviews and discussions.
- All the ambiguities and contradictions are identified and resolved.

Requirements specification

- User requirements are captured and organized into a Software Requirements Specification (SRS) document.
- The main components of SRS are functional requirements, non-functional requirements and goals for implementation.
- Very little information on top-level analysis and design is also included in this document.

Facilitator Notes:

Explain to participants, how in this phase, the expectations and goals of the project are defined, and risks are analyzed.

The aim of the requirements analysis and specification phase is to understand the exact requirements of the customer and to document them properly. This phase consists of two distinct activities:

- Requirements gathering and analysis
- Requirements specification

The aim of 'requirements gathering' is to collect all relevant information from the customer regarding the product to be developed. Once this is done and clear understanding of the customer requirements is available, then the incompleteness and inconsistencies are removed. The requirements analysis activity is started by collecting all the relevant data related to the product to be developed from the users of the product and from the customer through interviews and discussions.

For example, to do the analysis of a business accounting software required by an organization, the analyst should interview all the accountants of the organization to assess their requirements and expectations. It is necessary to identify all ambiguities and contradictions in the requirements as there is a possibility that the data collected may contain several contradictions and ambiguities, since each user typically has only a partial and incomplete view of the system. Once all ambiguities, inconsistencies, and incompleteness have been resolved and all the requirements/expectations have been properly understood, the requirements specification activity can start.

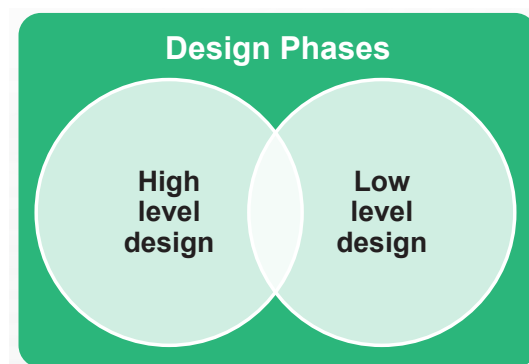
This activity includes the steps to systematically capture and organize the user requirements into a Software/hardware Requirements Specification (SRS) document. The customer requirements identified during the requirements gathering and analysis activity are organized into an SRS document. The important components of this document are functional requirements, the nonfunctional requirements, and the goals of the implementation.

1.8.3 Third Stage – Design Phase

The following activities are covered during the design phase:

- Creation of system design as per the requirements gathered during the previous phase, i.e., based on SRS document
- Functionality of hardware and software are separated out and the software modules are designed
- Definition of overall system architecture
- Documentation of design

There are two levels in the design phase:



Facilitator Notes:

Explain to participants, how a blueprint is drawn up for the developers along with a plan of meeting requirements.

The goal of the design phase is to transform the requirements specified in the SRS document into a structure that is suitable for implementation in some programming language. In technical terms, during the design phase the software architecture is derived from the SRS document.

The two phases of design are as follows:

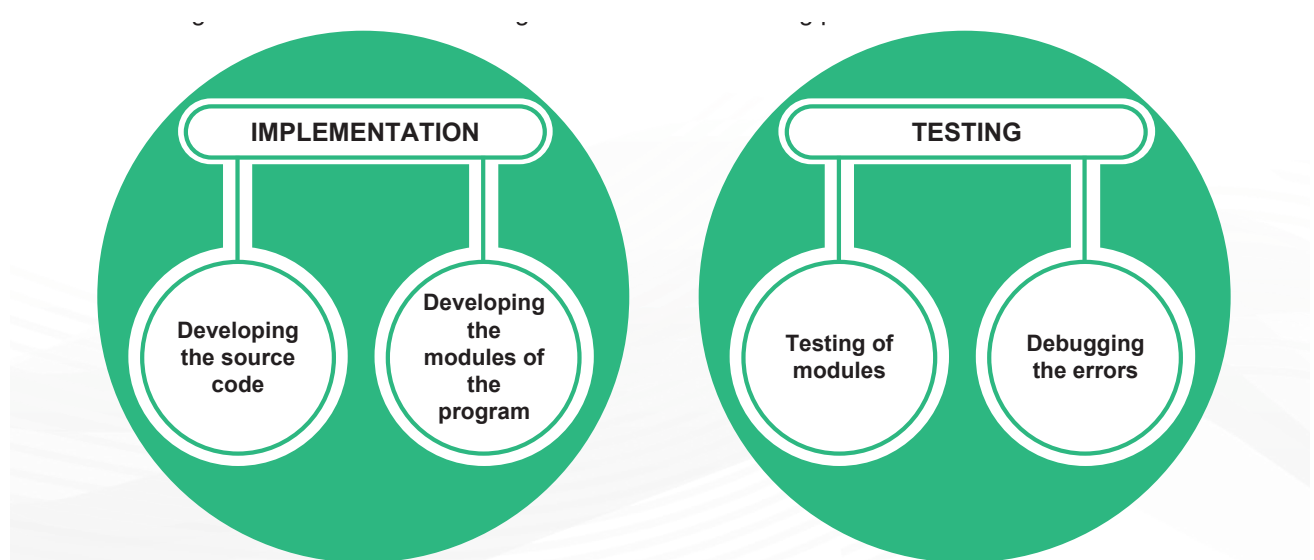
- **High level design:** High-Level Design (HLD) involves decomposing a system into modules, and representing the interfaces and invoking relationships among modules. An HLD is referred to as software architecture. A HLD document will usually include a high-level architecture diagram

depicting the components, interfaces, and networks that need to be further specified or developed. The document may also depict or otherwise refer to workflows and/or data flows between component systems.

- **Low level design:** LLD, also known as a detailed design, is used to design internals of the individual modules identified during HLD, i.e., data structures and algorithms of the modules are designed and documented. Program specifications are covered under LLD. The LLD describes each and every module in an elaborate manner so that the programmer can directly code the program based on it. There will be at least one document for each module. The LLD will contain:
 - a detailed functional logic of the module in pseudocode
 - database tables with all elements, including their type and size
 - all interface details with complete API references (both requests and responses)
 - all dependency issues
 - error message listings
 - complete inputs and outputs for a module

1.8.4 Fourth Stage – Coding and Unit Testing

The following activities are covered during the code and unit testing phase:



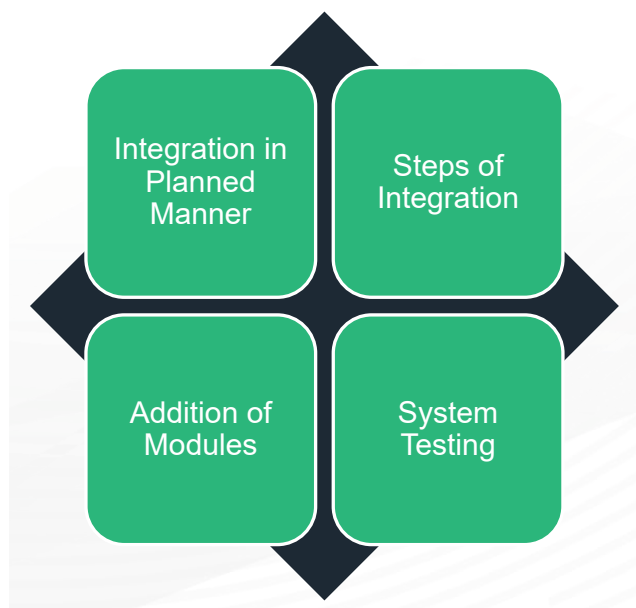
Facilitator Notes:

Let the participants know that this phase is called as development, implementation or coding and how the development takes place.

This stage is also called the implementation phase. This stage involves the coding and unit testing of software development and to translate the software design into source code. Each component of the design is applied as a program module. The end-product of this phase is a set of program modules that have been being tested individually to ensure the correct working of all the individual modules. The testing of each module in isolation is the most ideal way to debug the errors at this stage.

1.8.5 Fifth Stage – Integration and System Testing

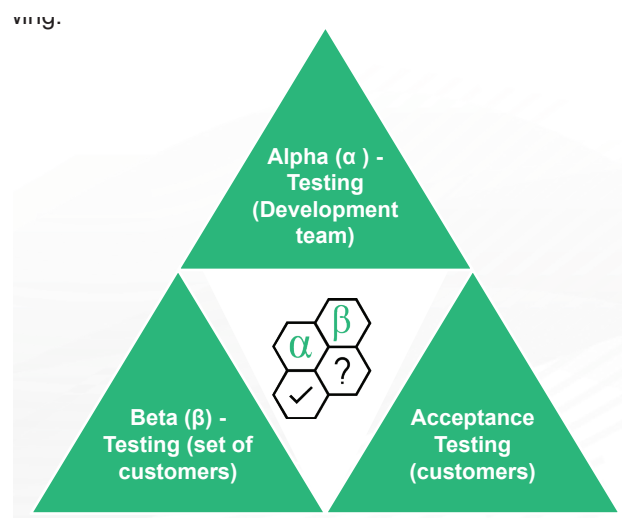
The following activities are covered during the integration and system testing phase:

**Facilitator Notes:**

Emphasize the importance of this stage to participants and how it ensures that the product meets the client's requirements.

Once the modules have been coded and unit tested, integration of different modules is undertaken. During the integration and system testing phase, the modules are integrated in a planned manner. The different modules making up a software product are almost never integrated in one shot. Integration is normally carried out progressively over a number of steps. During each integration step, the partially integrated system is tested and a set of previously planned modules is added to it. Finally, when all the modules have been successfully integrated and tested, system testing is carried out to ensure that the developed system conforms to its requirements laid out in the SRS document.

This is a level of software testing where a complete and integrated software is tested. The various tests may include the following:



Facilitator Notes:

Explain the participants about the different categories of system tests.

System testing usually comprises of three different kinds of testing activities:

1. **α – testing:** It is the system testing carried out by the development team.
2. **β – testing:** It is the system testing carried by a friendly set of customers.
3. **Acceptance testing:** It is the system testing performed by the customer himself after the product delivery to determine whether to accept or reject the delivered product.

System testing is normally carried out in a planned manner according to the system test plan document. The system test plan identifies all testing related activities to be performed, mentioning the schedule of testing, and defining the resources. This step also includes all the test cases and the expected outputs for each test case. After integrating the unit tested code, it is made sure that it works well, as expected and error-free. All the functional and non-functional testing activities are performed to check whether the system meets the requirement perfectly. The progress on testing is tracked through tools like traceability metrics and ALM. Finally, the progress report of testing activities is prepared.

1.8.6 Sixth Stage – System Deployment and Maintenance

The following activities are covered during the system deployment and maintenance phase.

- Product Deployment
- Development and Maintenance

Maintenance

- Corrective Maintenance
- Perfective Maintenance
- Adaptive Maintenance



Facilitator Notes:

Explain to the participants that the product is implemented according to the agreed-upon requirements and one more round of testing and verification should be done after implementation.

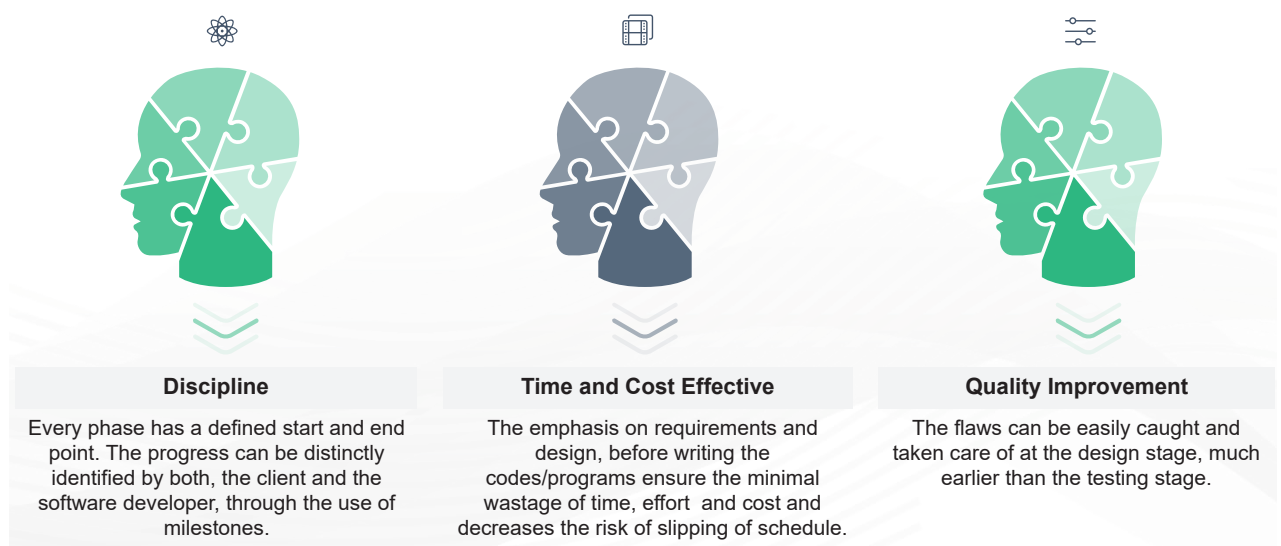
Once the functional and non-functional testing is completed, the product is deployed in the customer environment or released in the market.

Maintenance of any software product requires much effort than the efforts put in to develop the product. Researches confirm that the relative effort of development of a typical software product to its maintenance effort is roughly in the 40:60 ratio.

Maintenance involves performing any one or more of the following three kinds of activities:

- a) **Corrective maintenance:** Correcting errors which were left undiscovered during the product development phase is the part of maintenance
- b) **Perfective maintenance:** Improving the implementation of the system and enhancing the functionalities of the system according to the customer's requirements. This is done to alter attributes or improve performance.
- c) **Adaptive maintenance:** Porting the software to work in a new environment. For example, porting may be required for the smooth running of the software to work on a new computer system or with a new operating system. The defects uncovered during live use of the software are also taken care of in the maintenance stage.

1.9 Advantages of Waterfall Model



Facilitator Notes:

Explain the characteristics of Waterfall Model. There are many reasons why a waterfall model has been so popular over the years few are listed on the slide.

The central idea of the waterfall model is to spend the majority of time, money and effort up front: 20-40% in the first two phases, 30-40% on coding/development, and the rest during implementation and maintenance.

1. **Discipline:** Every phase has a defined start and end point. The progress can be distinctly identified by both, the client and the software developer, through the use of milestones.

2. **Time and cost effective:** The emphasis on requirements and design, before writing the codes/ programs ensures the minimal wastage of time, effort and cost and decreases the risk of slipping of schedule.
3. **Quality improvement:** The flaws can be easily caught and taken care of at the design stage, much earlier than the testing stage.

1.10 Shortcomings of the Waterfall Model

- ✓ Possibilities of errors
- ✓ Not suitable for all projects
- ✓ Implicit assumptions
- ✓ High risk and uncertainty
- ✓ Non-adaptive design constraints
- ✓ Ignores mid-process user/client feedback
- ✓ Delayed testing period
- ✓ Makes changes difficult



Facilitator Notes:

Explain to participants that new software development models have been introduced, since waterfall model had its own disadvantages. Discuss the shortfall of the waterfall model.

Shortcomings of the waterfall model

It is assumed that no developmental error is ever committed by the engineers during any of the life cycle phases in the classic waterfall model . However, in practical development environments, large number of errors are committed in almost every phase of the life cycle.

The source of the defects can be many, such as due to oversight, wrong assumptions, use of inappropriate technology, communication gap between the project engineers, etc. Following are the disadvantages of the waterfall model:

- In waterfall model, bugs/errors are usually detected much later in the life cycle. For e.g., a defect might have gone unnoticed till the coding or testing phase. Once a defect is detected, the engineers need to go back and rectify some of the work done during that phase and the subsequent phases. Therefore, in any practical software development work, it is not possible to strictly follow the classic waterfall model.
- This model is not good for those projects where the requirements keep changing.
- Implicit assumptions of that, the design can be translated into a product can lead to roadblock at a very late stage.
- Lack of adaptability across all stages of development is the most difficult disadvantage of the waterfall model. When a test in the fifth stage explains a fundamental flaw in the design of the system, it requires a dramatic leap backward in stages of the process. In the worst case, it leads to a devastating realization regarding the legitimacy of the entire system. While most experienced teams and developers would argue that such revelations shouldn't occur if the system was properly designed in the first place, not every possibility can be accounted for, especially when stages are so often delayed until the end of the process.
- Due to the strict incremental process that the waterfall model enforces, user or client feedback is received only during the later stages of the development cycle. While project managers can obviously enforce a process to step back to a previous stage due to an unforeseen requirement or change coming from a client, it will be both costly and time-consuming, for both the development team and the client.
- Waterfall strictly introduces testing quite late into the cycle. Most bugs or even design issues won't be discovered until very late in the process, but it also encourages poor coding practices that lack enthusiasm and determination, since testing is only an afterthought.
- Waterfall is based on steps that keeps the teams strictly moving in a forward direction, there's no room for unplanned changes or updates. If the team has strictly followed the waterfall model nearly to the end of the project, and faces a sudden and unexpected change in terms of goals or scope, pivoting will become mighty difficult. Much of the work done so far may go useless.

What did You Grasp?



1. Which model is also called as the classic life cycle or the Waterfall model?
A) Iterative Development
B) Linear Sequential Development
C) RAD Model
D) Incremental Development
2. What is the simplest model of software development paradigm?
A) V-model
B) Spiral model
C) Big Bang model
D) Waterfall model



3. In which of the following phases of maintenance, the software is ported to work in a new environment?
A) Corrective maintenance
B) Perfective maintenance
C) Adaptive maintenance
D) None of the above
4. Which of the following testing is carried out by a primary group of customers?
A) Alpha-testing
B) Beta-testing
C) Gamma-testing
D) Delta-testing

Facilitator Notes:

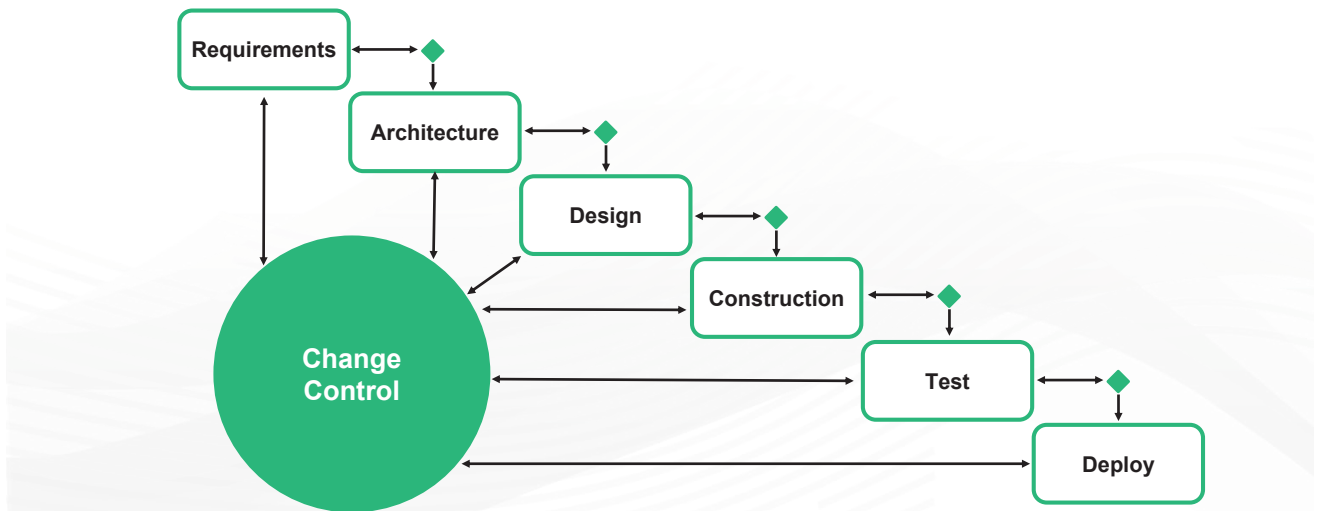
Tell the participants that they will be going through a knowledge check question.

Answers:

1. b. Linear Sequential Development
2. d. Waterfall model
3. c. Adaptive maintenance
4. b. Beta-testing

1.11 Gated Waterfall Model

The following picture depicts the phases in the gated waterfall model.



Facilitator Notes:

Explain the participants about the gated waterfall model, which is a modified version of the classic waterfall model. Explain the participants about the changes in the gated model, compared to the classical model.

In the gated waterfall model, the development has to pass through a 'quality gate' to move between phases. The quality gate is based on the review and acceptance of artifacts – for example a Software Requirements Specification (SRS) might be a result of the Requirements phase, a Software Architecture Design (SAD) is the result of the Architecture phase, and so on. At the end of each phase, the feedback goes back to any phase by means of a Change Control process. This change control process is considered to be a change prevention process in most traditional teams. Though there are quality gates, they do not have any major impact on the quality, in practice.

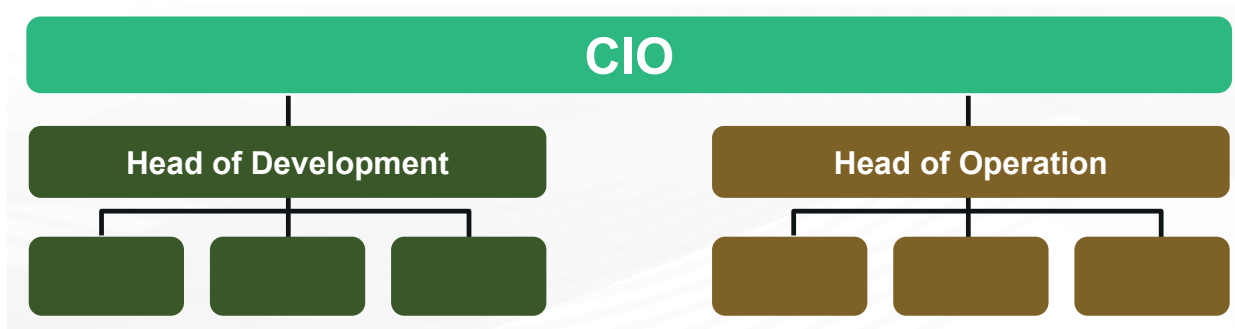
The major difference between the classical and gated waterfall model is the presence of quality gate between the phases, which provides room for feedback and corrections.

We learnt about the important traditional software development model. We'll now look at the issues with traditional development methodologies.

1.12 Traditional IT Organizations

Cultural hindrance between development and operations teams in traditional SDLC.

- Development and operations are two sides of an equation holding their own roles and responsibilities.
- The development team works independently on code. The code then sent to the testing team for validation.
- Operations team comes in toward the end of the process, handover of the release.
- There is no collaboration between these teams.



Facilitator Notes:

Explain how development and operations happen in a traditional SDLC, and how conflicts emerge out of it.

So far we have seen about software and its types, the history of software engineering and the waterfall model as an example for traditional software development.

Organizations that follow the traditional way of software development work with strict principles and in these organizations, the development and operations teams function as two separate entities. Development team tends to be driven by how many new functionalities can be churned out in a given time, therefore change is its incentive. Operations team on the other hand, tends to be driven by the stability of the status quo and its incentive is therefore resisting change.

There exists a cultural hindrance between development and operations teams in traditional SDLC. In a traditional setup, the development team works on code which is then sent to the testing team for validation against requirements. The operation team comes in toward the end of the process, where the handover of release is given.

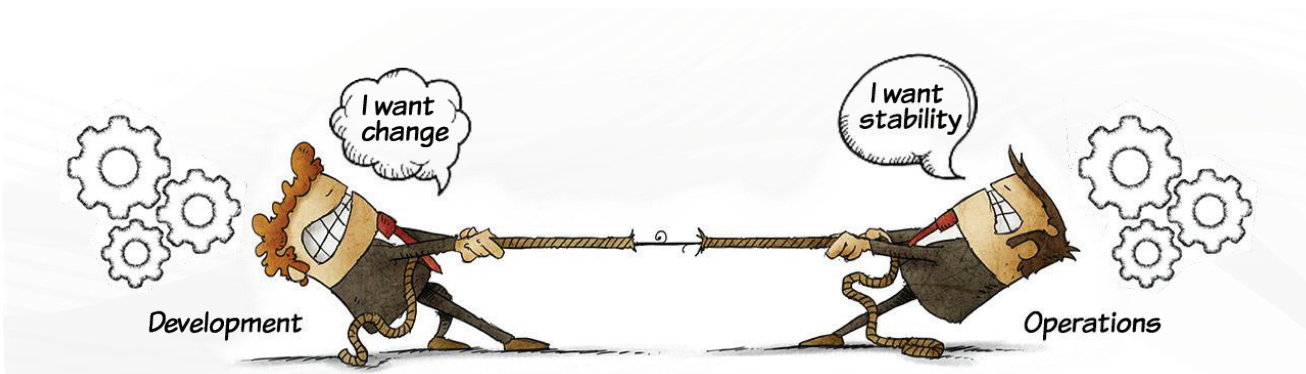
Most of the organizations, who have adopted traditional SDLC, face a situation like this on a daily basis:

- A developer produces some code and tests it in a pre-production environment.
- Operations then pushes the updated code into the production environment.
- Something breaks in the production environment, and the operations team reports a bug to the developer.
- The developer tests the bug in the pre-production environment and cannot reproduce it.
- The developer sends the bug back to operations, thinking it's an operational issue.
- The issue then goes back and forth between teams, wasting valuable time and creating the potential for end-user frustration.

This causes a disconnect and there is hardly any collaboration between the development and operations teams. This leads to the rise of conflicts among the teams, which has a direct impact on the software being developed and delivered to the customer. More about this will be explained in the forthcoming section.

1.13 Developers vs IT Operations Conflict

1. Meaning
2. Development Changes
3. Confusions /Lack Of Communication
4. Operations Stability



Facilitator Notes:

Explain that organizations are far from being a strategic, well-oiled discipline that directly delivers business value. Many challenges exist that hamper business operations, growth, and success. There is tension (or wall) between development and operations teams in software development circles.

Ask the following questions to the participants based on the images depicted on this slide:

- What is the key goal of the development team?
- What is the function of operations team?
- Is the development and the operations teams connected to each other?

Various challenges occur due to contracting goals of the various teams, especially the development and the operations, involved in the software development and delivery. The job of the development team is to build software and apply changes to incorporate new features and fulfil the internal as well as the external requirements. On the other hand, the operations team focuses on stability, reliability, and performance of the systems maintained by the team. The two competing contradicting goals of the two teams result in a wall of confusion.

The wall of confusion prevents required communication between the development and the operations teams and results in severe problems in production that causes blast like situations, such as no methodical handover to the operations team is done leading to 'half cooked meal' like situation. Consequently, the operations team faces problems in production that they are unable to solve and look back to the development team for resolution. This loopback delays

problem resolution.

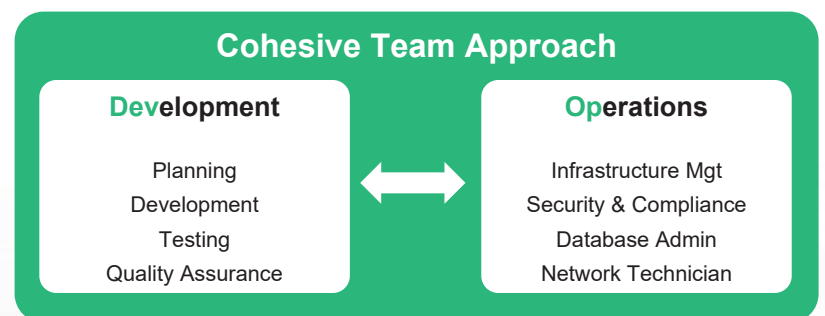
In the absence of required discussions between the development and the operations teams during earlier phases of development, a lot of useful information is not shared between the two teams. Such information is crucial for the operations team to get ready for the upcoming changes to the applications under development. For example, the operations team can share valuable information from their experience of managing production environment. This information can help the development team design and develop robust applications. However, due to lack of communication between the two teams, this information sharing is missed.

A critical part of transition between the development and the operations teams is knowledge articles. These articles help the operations team solve known problems. In the presence of the wall of confusion, these elaborate knowledge articles are missing. As a result, the operations team takes extra time to solve trivial problems.

The wall of confusion is caused due to the conflicting motivations and mindset with regard to the development and operations of the software activity. This disconnect results in conflict and inefficiency which is generally a mindset called wall of confusion.

1.14 Problems with the Traditional Development and the Operations

- Organizational silos
- Different mindsets
- Different implementations
- Different tools
- Lack of interest in learning other tools
- Different environments
- Loss of work
- Blame game
- Build rollback



- Disintegrated process
- No feedback loop

Facilitator Notes:

The slide lists some of the problems between the development and the operations teams due to the traditional way of developing applications.

Some of the problems with the traditional development and the operations teams include:

- **Organizational Silos:** The two teams work in isolation that do not allow them to understand each other's problems and perspectives. Every individual and every single team, whether it's development, operations, quality assurance, or the support team, all come across challenges on a daily basis. Regardless of who is responsible, the problem needs to be solved. Without collaboration, this process takes longer and can create further problems that may not be immediately apparent. Working together and communicating efficiently allows you to implement solutions that will help prevent similar incidents in the future.
- **Different Mindsets:** The development team always wants to incorporate every new technique/feature to do their work efficiently. On the other hand, changes are not at all acceptable by the operations team because change results in instability. Therefore, change is the biggest enemy for the operations team.
- **Different Implementations:** The different implementations to perform the same work by the team's results in incompatibility and lead to various bugs in the QA and the production environments.
- **Different Tools:** The different tools used by the two teams lead to various errors and bugs in the production environment. For example, the development team might deploy to a test environment using the dependency management tool, while operations team might use a home-grown script for the process.
- **Lack of Interest in Learning Other Tools:** Each team considers its tool or style of working to be the best and does not want to learn a new tool.
- **Different Environments:** The different environments, such as development, production, and testing, is one of the biggest causes of various errors and bugs raised by the different teams.
- **Loss of Work:** The various errors and bugs results in loss of valuable efforts.
- **Blame Game:** A lot of differences between the teams and environments force the teams to pass on the blame of delayed delivery or build rollback on each other. It's unnecessary placing blame and pointing fingers, the key here is to use the available time and resources to solve the issue.
- **Build Rollback:** A build is a version of the software that is deployed and rolled out to the customer after stringent tests. During the build process, source code is converted into a standalone software

artifact, called build artifacts that can be run on computer systems. A build doesn't always go right. Many times teams will be forced to roll back the build due to various reasons, such as incorrect client requirements, incorrect database (DB) in the QA or production environment, incompatible tools and others.

- **Disintegrated Processes:** Development processes do not integrate well with operations processes.
- **No Feedback Loop:** Lack of a continuous feedback loop in development and operational processes causes gaps.

To solve the issues that arise due to the wall of confusion and the conflicts that exists between development and operations teams, in traditional IT organizations, and to develop quality software in a short time, companies have started moving from traditional approaches to DevOps.

What is the solution?

The wall of confusion between the development and operations teams is one of the major reasons for the emergence of DevOps. With DevOps, the development and operations teams work in collaboration to minimize the effort and risk involved in releasing software. Collaboration can be ensured by the operations team by means of giving constant feedback to the development team about the code, analyzing the impact considering end users and troubleshooting any problems together to gain stability of the product. DevOps enables a cultural change to remove the barrier between development and operations, working together for common set of objectives. Some of the approaches for solving these issues is:

- Working as cohesive teams
- Having shared objectives
- Coordinating work and sharing information between teams
- Collaboration
- Use of shared tools

DevOps is fundamentally an extension to Agile and Lean principles, as it attempts to instil those same values and practices into Operations. You will learn in detail about Agile development in the forthcoming modules.

What did You Grasp?



1. There is a negative effect on deployment and delivery dates in case of conflict between the developers and the IT operations.
A) True
B) False
2. Which of the following is NOT a consequence of wall of confusion?
A) Lack of communication
B) Errors and bugs due to the use of different tools
C) Application delivery in a fast pace
D) No methodical handover

Facilitator Notes:

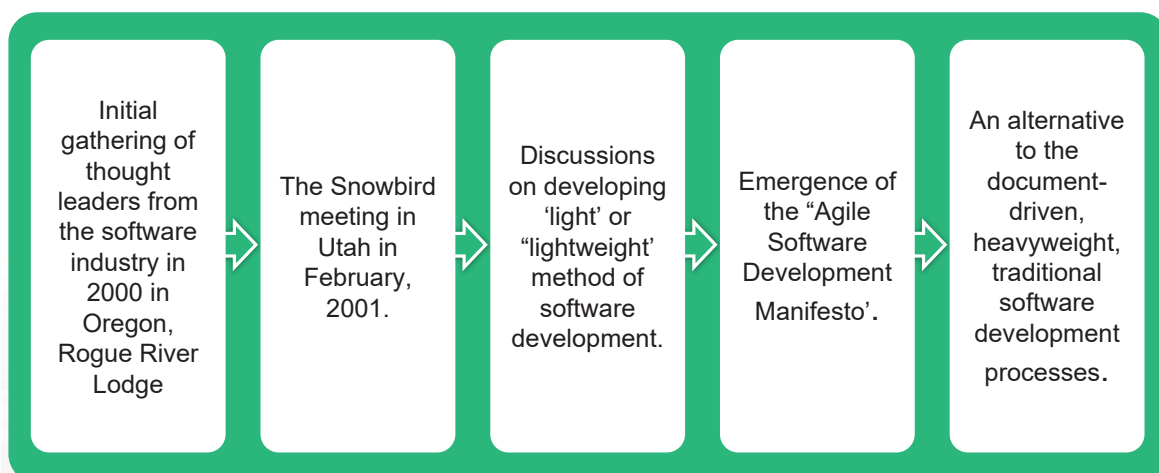
Tell the participants that they will be going through a knowledge check question.

Answers:

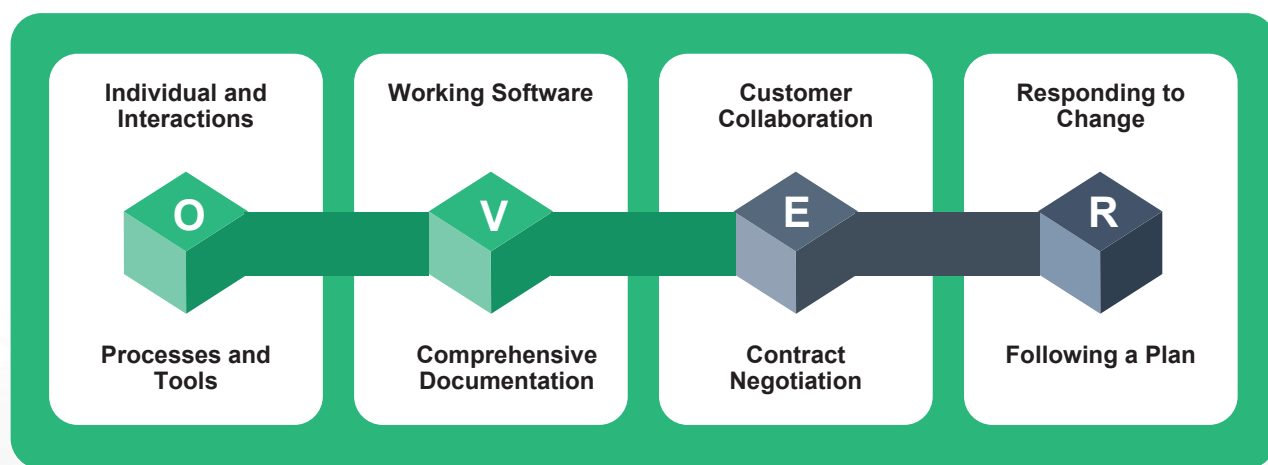
1. a. True
2. c. Application delivery in a fast pace.

1.15 Birth of Agile

The following is the process illustrates the birth of Agile:



1.16 Four Values of the Agile Manifesto



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

Facilitator Notes:

- Show the participants the values of the Agile Manifesto
- Describe the importance of values

The agile manifesto reads:

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

“Individuals and interactions over processes and tools.

Working software over comprehensive documentation.

Customer collaboration over contract negotiation.

Responding to change over following a plan.”

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

Each of these four values is described in the upcoming sections.

1.16.1 Agile Manifesto

Agile Manifesto Value 1: Individual and team interactions over processes and tools

INDIVIDUALS AND INTERACTIONS

- Communication—the key to the success of a project
- People build Products
- Have the focus on people and the source of energy
- Self-organization of cross-functional teams—to identify scope, negotiate, accept, define, collaborate, share, and solve problems
- Individuals need to be motivated
- Interactions need to be fostered among team members, customers and other stakeholders

Facilitator Notes:

Inform the participants that the first Agile Manifesto value talks about individuals and interactions.

Prioritizing individuals and interactions over processes and tools is the first value of the Agile manifesto. People drive the development process since they are the ones who respond to changing business needs and develop processes and tools in response to change.

If a team prioritizes processes, technology or tools, then the individuals in the team become less responsive and meeting the customer requirements become a difficult task.

Communication between the team members, customers and other stakeholders is critical for understanding the business requirements and delivering value. If individuals are valued over processes and tools, the communication becomes fluid and the interactions happen, based on the change in business requirements. If processes are valued more, the interactions become scheduled and less adaptive to changes. There is a possibility of the customer's requirements losing importance because of the stringent schedules.

Agile Manifesto Value 2: Working software over comprehensive documentation

WORKING SOFTWARE

- Create and deliver value
- Customer satisfaction is important
- Deliver frequently and consistently; offer business value to the customer
- The primary goal of software development is to create software, not lengthy documents
- Write documentation that adds value
- Deliver what the customer wants; documentation is always supplementary
- Customer-focus is the primary focus

Facilitator Notes:

Tell the participants that you will be covering the second value here, which is working software.

The second value as per the agile manifesto is a working software over comprehensive documentation.

Jim Highsmith, one of the authors of the Agile Manifesto and the primary developer of the 'Adaptive Software Development' Agile Method, says, "We want to restore a balance. We embrace modelling, but not in order to file some diagram in a dusty corporate repository. We embrace documentation, but not hundreds of pages of never-maintained and rarely-used tomes. We plan, but recognize the limits of planning in a turbulent environment."

If we give customers a choice of selecting between comprehensive documentation and working software, they would prefer the latter. Agile focuses more on building the product to satisfy the needs of the customer, than to giving them pages and pages of documentation.

Writing pages and pages of documentation for developing the product, consumes an enormous amount of time, hence longer time periods between documentation and delivery. A lot of time, money and energy is spent in writing technical and functional specifications, customer's business requirements, user interface specifications, design documents, documents on testing and much more. And the best part is that approvals are required for each and every document written by people, at multiple levels. This is the major cause of the delay in the development and delivery of the actual product.

In agreement with Jim's statement, Agile does not rule out the need for documentation. Agile emphasizes that the document is streamlined in a way that it gives the exact picture to the developer of what is exactly needed to build the software, without getting lost in the intricacies.

According to Agile methodologies, the requirements are documented as user stories, so the developer gets clarity on the business requirements. This eventually becomes a guide for the developer to start building the exact functionalities. Agile does give importance to documentation, but a working software is more critical from a customer's viewpoint than documentation. Agile also emphasizes that delivering working pieces of the software at frequent intervals matters the most to the customer.

Agile Manifesto Value 3: Customer collaboration over contract negotiation

CUSTOMER COLLABORATION

- Flexibility and co-operation in terms of customer's needs
- Work with the customer
- Make sure the intent of the contract is satisfied
- Understand customer's product vision by close collaboration
- Let the contracting models be flexible
- Maintain relationships

Facilitator Notes:

Tell the participants that you will be talking about customer collaboration next.

Customer collaboration over contract negotiation is the third value described in the agile manifesto.

Negotiation enables the customer and the product manager to work out the details of product delivery. There is a room for re-negotiation as well. In traditional software development processes, customer engagement is more before the start of the development process and after the product is completely ready. There are two major disadvantages to this. The customer gives the complete specifications/requirements in great detail, well before the development starts. There is a chance for confusions and multiple rounds of discussions happening even before the actual work starts, which causes the delay in the development. If the product is

completely ready, and the customer starts changing the requirements, much effort will be involved in redoing things.

Agile manifesto emphasizes that the customer is engaged in and collaborates throughout the development process. Demo sessions should happen periodically. This enables feedback sharing at regular intervals and the product gets developed iteratively. The development team and the customer can make sure that the requirements are met at each and every point in time during the development process.

This value suggests the end - user being part of the development process, attending all the meetings and making sure changes are communicated then and there and ensuring the smooth delivery of the project.

Agile Manifesto Value 4: Responding to change over following a plan

RESPONDING TO CHANGE

- Change is the reality, the worst enemy of any plan
- Changes in customer's business needs - direct impact on developer's plans
- Adaptive planning for accepting change
- Change to be reflected in the product
- Show improvement

Facilitator Notes:

Tell the participants that you will be talking about responding to change which is the next value of the Agile Manifesto.

Responding to change over following a plan is the fourth value as per the Agile Manifesto.

A change was considered expensive and avoidable by the traditional software development methodologies. Planning was considered much more important than responding to changes. The intention was to have the detailed and comprehensive plans, where feature sets and functionalities are pre-defined. Since traditional methods follow a sequential order in the development and delivery of software, a lot of unnecessary dependencies arise. High priority is given to each and every step of the development process.

On the other hand, Agile suggests shorter iterations. This enables change in priorities from iteration to iteration, according to the customer's requirements. There is room for the addition of new features in the next iteration, without having any dependency on the previous iteration. Agile views change as a means for improvement and believes that a change adds value to the product.

A process called Method Tailoring describes the Agile approach towards change. An Agile Information Systems Development Method defines Method Tailoring as a process or capability in which human agents determine a system development approach for a specific project situation through responsive changes in, and dynamic interplays between contexts, intentions, and method fragments.

With Agile, the process modifications that fit the team are allowed.

1.16.2 Twelve Principles of the Agile Manifesto

- 1 Our highest priority is to satisfy the customer through an early and continuous delivery of valuable software.
- 2 Welcome changing requirements, even late in the 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 a 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.

Facilitator Notes:

Explain the 12 principles in detail.

1.17 Scrum: An Introduction

Following are the key details of Scrum:

- Scrum is one of the most popular and widely used frameworks to implement Agile in industry projects.
- Scrum has gained so much popularity that it is used synonymously with Agile, but in reality they are different.
- Scrum is a framework used for incremental product development by deploying one or more cross-functional, self-organizing teams.
- Using scrum, product development is carried out in small, fixed-length iterations, called sprints.
- Sprints are not longer than 30 days, short sprints are preferred. Every sprint is intended to build a potentially shippable product increment.

Facilitator Notes:

Give the participants an overview of Scrum. Give them a recap of the evolution of Agile and walk them through the history of scrum.

Scrum is one of the most popular agile methodologies. Scrum was introduced by Jeff Sutherland and Ken Schwaber. Ken and Jeff co-presented the Scrum concept at the 1995 OOPSLA (Object-Oriented Programming, Systems, Languages & Applications) conference. This gave the first, formal public definition of Scrum and it was improvised and redefined by many industry experts. Scrum is so popular that it is widely used as a synonym for Agile itself, where in reality these two are different. Scrum is one of the ways of implementing Agile. Ken himself stated once that by early 2009, about 84% of the organizations who were using Agile in their projects were using Scrum.

Scrum is primarily used for incremental product development. Scrum deploys one or more self-organizing cross-functional teams. In scrum, the product development is carried out in small, fixed-length iterations, called sprints. Any sprint should ideally take not more than 30 days, shorter sprints are generally preferred. At the end of every sprint, a releasable, i.e., a potentially shippable product increment is ready.

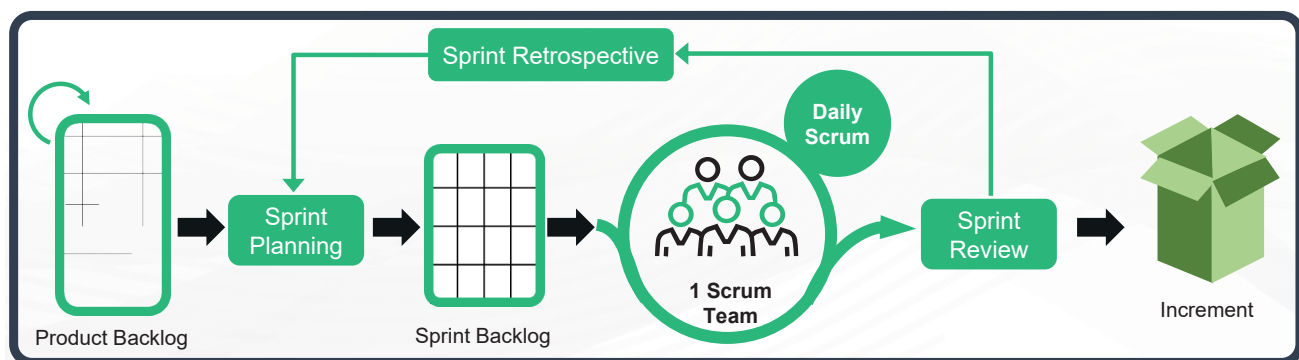
According to the developers of Scrum, it is:

- Lightweight
- Simple to understand
- Difficult to master

1.17.1 Why Scrum?

- Scrum is an alternative to the traditional waterfall method.
- Where traditional approaches depends on complete understanding of the requirements before starting off, scrum includes all the development activities in each of the sprints.
- Scrum focuses on developing high-value features first with the frequent incorporation of feedback.

The following figure shows the overall scrum framework.



Source: *Scrum.org*.

Facilitator Notes:

Explain the participants why Scrum is needed.

Scrum has been extensively used to develop not only software, but also hardware, embedded software, networks of interacting function, autonomous vehicles, schools, government, marketing, managing the operation of organizations and most of the stuff that we use in our daily lives.

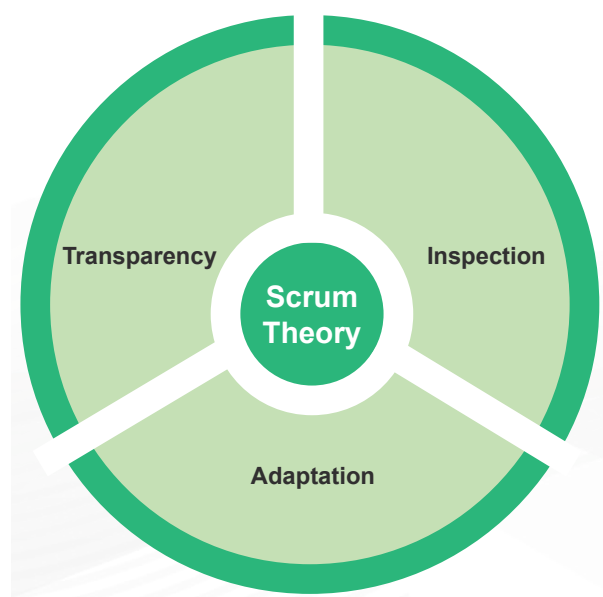
Scrum is a very good alternative to the traditional waterfall model. Traditional approach requires that the complete requirements are understood before the development commences and there might not be any room to accommodate changes specified. Scrum is always focused on developing high value features first, with frequent incorporation of feedback. The greatest potential benefit of Scrum is for complex work involving knowledge creation and collaboration, such as new product development. Scrum is usually associated with object-oriented software development. Its use has also spread to the development of products such as semiconductors, mortgages, and wheelchairs.

With the increase in technology, market, and environmental complexities and their interactions, the utility of Scrum in dealing with complexity is proven daily. We can see the efficiency of Scrum in iterative and incremental knowledge transfer. Scrum has now been widely used for products, services, and the management of the parent organization.

Scrum is effective with a small team of people, who are highly flexible and adaptive. The same effect can be seen with single or multiple teams. Teams collaborate among themselves using sophisticated development techniques and target release environments.

1.17.2 Scrum Theory

Scrum is based on empirical process control theory, called empiricism. This ensures that knowledge originates from experience and decisions are taken based on known things. The picture illustrates the three pillars of scrum theory.



Facilitator Notes:

Explain the participants about the three pillars of scrum theory.

Scrum is based empirical process control theory, also called as empiricism. This ensures that knowledge originates from experience and decisions are made using known things.

Scrum follows an iterative and incremental approach for optimizing predictability and controlling risk. The three pillars of the scrum theory are as follows.

Transparency: Transparency ensures that the important aspects of the processes related to any business are visible to those who are responsible for the outcome. It is important that these aspects are defined using a common standard. This ensures that all the observers understand the aspects in the same way. An example for this is that the definition of 'Done' is common for those who do the work and those who evaluate the results of the work.

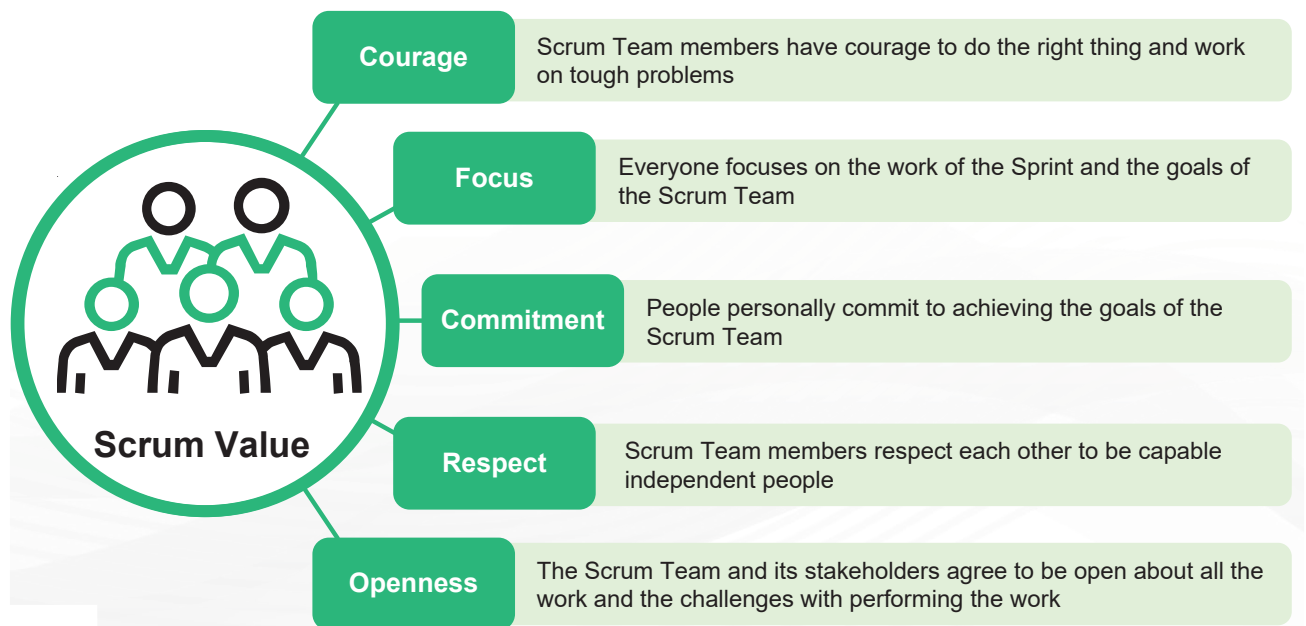
Inspection: In order to detect variances that may affect the process, Scrum users should inspect the Scrum artifacts and progress at frequent intervals. They should also make sure that inspection doesn't affect the actual workflow. Suitably experienced inspectors should carry out the inspection process to derive value out of it.

Adaptation: If an inspector identifies that any deviation in any of the process aspects makes the product unacceptable, adjustments in that process should be made. This adjustment should be done as soon as the defective process is identified, so that further deviations can be avoided.

Four formal events have been specified by Scrum for inspection and adaptation, which are described in a later section:

- Sprint Planning
- Daily Scrum
- Sprint Review
- Sprint Retrospective

1.17.3 Scrum Values



Source: [scrum.org](https://www.scrum.org)

Facilitator Notes:

Explain the participants about the five values of Scrum. There is a five-blog series published on the Scrum website, that will help participants learn more about Scrum values and the examples. Ask the participants to go through these blogs and help them understand the values better.

- Part 1 – Focus - <https://www.scrum.org/resources/blog/maximize-scrum-scrum-values-focus-part-1-5>
- Part 2 – Openness - <https://www.scrum.org/resources/blog/maximize-scrum-scrum-values-openness-part-2-5>
- Part 3 – Courage - <https://www.scrum.org/resources/blog/maximize-scrum-scrum-values-courage-part-3-5>
- Part 4 - <https://www.scrum.org/resources/blog/maximize-scrum-scrum-values-commitment-part-4-5>
- Part 5 – Respect - <https://www.scrum.org/resources/blog/maximize-scrum-scrum-values-respect-part-5-5>

Scrum values have been defined to ensure that the three pillars of Scrum, i.e., transparency, adaptation and inspection, make sense for all the Scrum users. Five values have been defined by Scrum as follows:

- Commitment: People should be committed to achieving the goals.
- Courage: The Scrum Team members have courage to do the right thing and work on tough problems.
- Focus: Everyone focuses on the work of the Sprint and the goals of the Scrum Team.
- Openness: The Scrum Team and its stakeholders agree to be open about all the work and the challenges with performing the work.
- Respect: Scrum Team members respect each other to be capable, independent people.

Scrum is said to be successfully implemented only if people become more proficient in these five values.

Go through the following blog series to understand Scrum Values along with examples

- Part 1 – Focus - <https://www.scrum.org/resources/blog/maximize-scrum-scrum-values-focus-part-1-5>
- Part 2 – Openness - <https://www.scrum.org/resources/blog/maximize-scrum-scrum-values-openness-part-2-5>
- Part 3 – Courage - <https://www.scrum.org/resources/blog/maximize-scrum-scrum-values-courage-part-3-5>
- Part 4 - <https://www.scrum.org/resources/blog/maximize-scrum-scrum-values-commitment-part-4-5>
- Part 5 – Respect - <https://www.scrum.org/resources/blog/maximize-scrum-scrum-values-respect-part-5-5>

What did You Grasp?



1. What are the iterations in Scrum called?
 - A) Release
 - B) Sprint
 - C) Release cycles
 - D) Build
2. Which of the following aspects of the Scrum theory ensures that every stakeholder understands the aspects in the same way?
 - A) Transparency
 - B) Inspection
 - C) Adaptation
 - D) Commitment

Facilitator Notes:

Tell the participants that they will be going through a knowledge check question.

Answers:

1. b. Sprint
2. a. Transparency

1.18 Scrum Roles

The following picture illustrates the three scrum roles in comparison with the traditional approach.

Product Owner The Holder of Product Value	Determines what needs to be done and sets the priorities to deliver the highest value Traditional approach: Controls the work
Scrum Master The Servant Leader	Protecting the Scrum process and preventing distractions Traditional approach: No equivalent
Development Team The Self-Organizing Group	Takes on and determines how to deliver chunks of work in frequent increments Traditional approach: Gets told what to do by the project manager

Source: Scrum Alliance

Facilitator Notes:

Give the participants an overview of the three scrum roles.

There are three scrum roles. They are as follows:

Scrum Master: The Scrum Master has to make sure that the performance of the Scrum Team is at their highest level. The Scrum Master also protects the team from both internal and external distractions.

Product Owner: The key responsibilities of the Product Owner are to maintain the product backlog, and to make sure that everyone is aware of their priorities and all the stakeholders are satisfied

The Development Team: The Development Teams are structured, self-organizing ones and manage their work on their own. The teams should be in synergy so that there is efficiency and effectiveness overall.

1.18.1 Scrum Master

The Scrum Master is responsible for promoting and supporting Scrum as defined in the Scrum Guide.

It is the responsibility of the Scrum Master to support the product owner, development team and the organization.

The following are the qualities of a Scrum Master, who has a leadership role:

- Works with organizations in implementing Scrum
- Makes sure that Scrum is understood and implemented
- Makes the environment workable, that also ensures team self-organization
- Guards the team from external interference and distractions and ensures harmony
- Helps to encourage improved engineering practices
- Doesn't manage the team
- Helps overcome hurdles

Facilitator Notes:

Explain the participants about the roles and responsibilities of a Scrum Master.

The Scrum Master is responsible for promoting and supporting Scrum as defined in the Scrum Guide. This is achieved by Scrum Masters by helping the Scrum teams understand Scrum theory, practices, rules, and values.

The Scrum Master leads and supports the Scrum Team. The Scrum Master also helps the people outside the Scrum Team how they can better interact with the team to produce maximum value.

Scrum Master's Service to the Product Owner

The Scrum Master offers service to the Product Owner in the following ways. These include, but not limited to:

- Makes sure that all the members of the Scrum Team understand well the goals, scope, and product domain
- Techniques to be implemented for an effective product backlog management
- Helping the Scrum Team understand about Product Backlog Items (PBIs)
- Helps the Product Owner do the product planning
- Makes sure that the Product Owner arranges the Product Backlog in a way that maximizes value
- Understanding and practicing agility
- Helps the Product Owner by facilitating Scrum events as per the requirements

Scrum Master's Service to the Development Team

The Scrum Master offers the following services to the Development Team, including, but not limited to:

- Enables the Development Team to self-organize and teaches them cross-functionality

- Supporting the Development Team in creating products that create true value
- Helps the Development Team overcome hurdles
- Facilitation of Scrum events
- Offering coaching to the Development Team from organizations that have not adopted Scrum, to understand the Scrum practices

Scrum Master's Service to the Organization

The Scrum Master serves the organization in ways, that include the following:

- Helping the organizations understand and adopt Scrum
- Planning and implementation of Scrum within an organization
- Helping employees and stakeholders understand and enact Scrum and empirical product development
- Helping the Scrum Team in increasing productivity
- Collaborating with other Scrum Masters to increase the effectiveness of the application of Scrum in the organization

1.18.2 Product Owner

Product Owner holds the complete responsibility of the product and is called Product Value Maximizer. Product owners understand the business and market requirements and plan the work to be done by the development team.

The responsibilities and qualities of a product owner include:

- Building and managing the product backlog
- Works closely with the organization and the Team to make sure everyone understands the work items in the product backlog
- Product Owner takes the sole responsibility of maximizing the return on investment (ROI) of the development effort
- Defines and constantly re-prioritizes the product backlog, makes adjustments to long-term expectations such as release plans
- Decides the set of requirements that the development team has to work on
- Remains accountable for the work done by the development team on product backlog
- Decides when to ship the product, by also taking into account frequent delivery

Facilitator Notes:

Explain the participants the roles and responsibilities of a Product Owner.

The Product Owner is responsible for maximizing the value of the product resulting from work of the Development Team, hence is called Product Value Maximizer. This process may vary widely across organizations, Scrum Teams, and individuals. It is important to understand that Product Owners are not project managers who manage the status of the program. Their focus lies in ensuring that the Development Team delivers value to the business.

The Product Owner is the single responsible person for managing the Product Backlog. Product Backlog management includes:

- Identifying and expressing Product Backlog items
- Prioritizing the Product Backlog items to best achieve goals and missions
- Value optimization of the work done by the development team
- Ensuring the visibility, transparency, and clarity on the tasks to be performed by the Scrum Team
- Making sure that the Development Team understands the PBIs and the priority

Product Owner is the authority who can change the priorities of items in the Product Backlog. It is the responsibility of the team to respect the decisions taken by the Product Owner.

1.18.3 Scrum Development Team

Characteristics of a development team are as follows:

- Collaborative, cross-functional, i.e., people of varied roles, with varied skills, who carry out different functionalities
- The Development Team has the capabilities to self-organize/self-manage, with roles assigned internally
- Along with the Product Owner, the team plans sprints, one at a time
- Development Team holds the autonomous right to plan the increment
- Initial few sprints require the team to sit together and work
- Scrum is against moving people across or splitting them between teams
- Ideal number per team is 6 +/- 3 members

Facilitator Notes:

Explain the participants about the development team and its characteristics, roles and responsibilities.

The responsibility of the Development Team is to deliver releasable (tested) Increment of 'Done' product at the end of each Sprint. This increment is required during the Sprint Review. Members of the Development Team solely creates the increment.

The characteristics of the Development Team are as follows:

- As stated above Development Teams are self-organizing. Product Backlog is turned into potentially releasable Increments at their own discretion, and even the Scrum Master doesn't have any say on this.
- Teams are cross-functional, in the sense that a single team has all the skills necessary to create a product increment.
- There is no specific job title for the members in a Development Team, irrespective of their responsibilities.
- According to Scrum, there is no sub-team in a Development Team, irrespective of the tasks they perform, like architecture, testing, operations, business analysts, etc.
- Though the team members have specialized skills and focus areas, ultimately the Team as a whole is accountable for the project.

Size of the Development Team

The ideal size of the Development Team is 6 +/- 3 members, anything more or less will result in complexities. A Team with less than 3 members will have troubles in interaction, which in turn result in loss in productivity. There might be a skill deficit in very small teams, which will make the Team unable to deliver the potentially releasable Increment. Having more than 9 members will cause coordination issues. The Product Owner and Scrum Master roles are not included in this count unless they are also executing the work of the Sprint Backlog.

What did You Grasp?



1. Who is responsible for protecting the team from internal and external distractions?
 - A) Product Owner
 - B) Scrum Master
 - C) The team itself
 - D) None of the above
2. Who builds the product backlog?
 - A) Product Owner
 - B) Scrum Master
 - C) The Development Team
 - D) Individual team member

Facilitator Notes:

Tell the participants that they will be going through a knowledge check question.

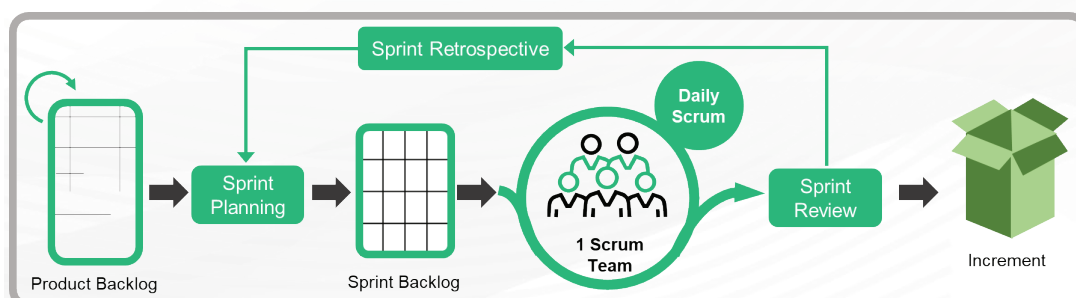
Answers:

1. b. Scrum Master
2. a. Product Owner

1.19 Scrum Sprints

Following are the key details of Scrum sprints:

- A sprint is a short, time-boxed period, during which a scrum team works and completes a certain amount of work.
- With scrum, any product is built in a series of sprints, one iteration of the product in each sprint.
- Sprints are used to break down big, complex projects into bite-sized pieces.
- Sprint is the heart of Scrum, during which a useable and potentially releasable product Increment is created.
- Sprints can be typically be of one week to one month in length and happen one right after the other to keep projects moving. Shorter Sprints are generally preferred.



Facilitator Notes:

Explain the participants about the sprints. Sprints are the important component of Scrum.

Sprint is the heart of Scrum, which is a time-box of one month or less. At the end of each Sprint, “Done”, i.e., a useable, and potentially releasable product Increment is created. The duration of Sprints are consistent throughout a development. A new Sprint starts immediately after the previous Sprint is complete.

What happens during a Sprint?

- Changes that may cause any danger to the Sprint Goal are not accommodated
- There will not be any compromise to quality goals
- With time and more learning, the Product Owner and the Development Team renegotiate and clarify the scope
- Any Sprint will be considered as a project, because a Sprint produces a useable version of the product
- Each Sprint has a goal as to what has to be built, a design and flexible plan that will guide, building it, the work, and the resultant product increment

Sprints are limited to one calendar month. Longer sprints may result in change in the definition of what is being built may change, with complexities and increased risk. Inspection and adaptation of the progress towards a Sprint Goal is done at least every calendar month.

Sprint Cancellation

Product Owner has the authority cancel the sprint before the time-box is over. Product Owner can also cancel a Sprint based on suggestions from stakeholders, Development Team or the Scrum Master.

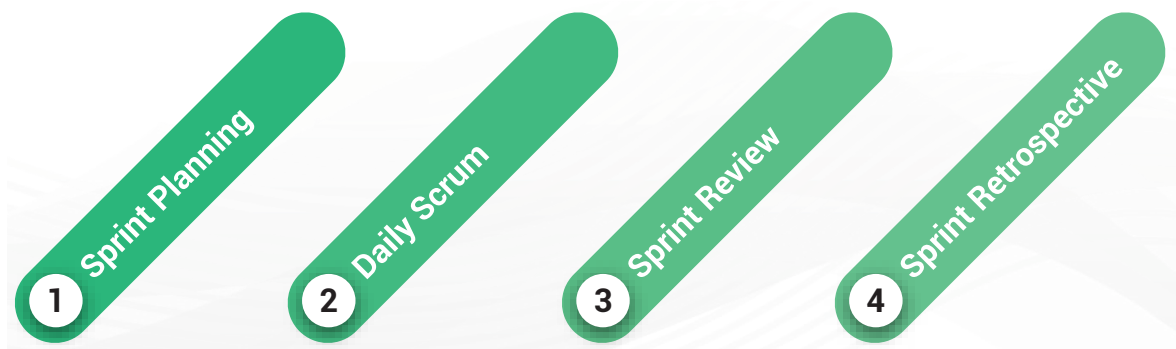
A Sprint would normally be cancelled when the goal becomes outdated. This might be because of business decisions or because of change in technology or market conditions. Because, Sprints are of short duration, cancellation might not make sense. Any completed and ‘Done’ PBIs are reviewed at Sprint cancellation. After the review, incomplete PBIs are re-estimated

and put back into the Product Backlog.

Sprint cancellations generally consume more resources, since the Sprint Planning has to start from scratch. Cancellations will visibly create an impact on the Scrum Team and these are very uncommon.

1.20 Scrum Ceremonies or Events

Events are described in Scrum for creating regularity and to reduce the need for meetings that are not described in Scrum. The picture illustrates the four major Scrum events:



Facilitator Notes:

Give the participants an overview of Scrum events, which are also called Scrum ceremonies.

Sprints contain and consist of four major events:

- Sprint Planning
- Daily Scrum
- Sprint Review
- Sprint Retrospective

We'll see each of these in detail in the upcoming slides.

1.20.1 Sprint Planning

Following are the key details of Sprint planning:

- At the beginning of each Sprint, the Product Owner and team hold a Sprint Planning Meeting to negotiate which Product Backlog Items (PBIs) will be converted to a working product during the Sprint.
- Scrum Master ensures that planning happens and the team understands the responsibilities.
- The Product Owner orders the product backlog based on business and technical requirements.
- The Development Team finalizes the amount of work that can be accomplished at the end of a Sprint.
- During the Sprint Planning Meeting, the team decides the ways of accomplishing the work.
- The time-box of Sprint Planning for a 30-day Sprint is eight hours, and is reduced proportionally for a shorter Sprint.

Facilitator Notes:

Explain the participants about Sprint planning meeting.

During Sprint Planning, the work to be done during a Sprint is planned and finalized. The Sprint Goal is also finalized during the sprint planning meeting. This plan is devised collaboratively by the Scrum Team. For a one-month Sprint, the Sprint Planning is time-boxed for 8 hours, this will reduce proportionally for shorter Sprints.

It is the responsibility of the Scrum Master to ensure that Sprint Planning happens and the team understands its purpose. Scrum Master also ensures that Sprint Planning happens within the specified time limit.

Sprint Planning answers two major questions:

- What can be delivered in the Increment resulting from the upcoming Sprint?
- How will the work needed to deliver the Increment be achieved?

Sprint Goal

The Sprint Goal is defined as the objective set for the Sprint that can be met through the implementation of the Product Backlog. With Sprint Goal, the Development Team Understands the purpose of building the Increment. The Goal also gives the Team the flexibility with respect to the functionality to be implemented within the Sprint. The Sprint Goal also determines the PBIs to be considered for that Sprint and they deliver one coherent function.

The Development Team always works towards achieving the Sprint Goal. Suitable functionality and technology are implemented by the Team to achieve the Sprint Goal. In case the Team finds that the work is not going in the right direction, they collaborate with the Product Owner and negotiate the scope of the Product Backlog within the Sprint.

1.20.2 Daily Scrum

Following are the key details of daily Scrum:

- The Daily Scrum is a 15-minute time-boxed event for the Development Team, held every day of the Sprint.
- During the Daily Scrum, the development team plans the work for the next 24 hours. The team shares the work done on the previous day, what will be done that day and what are the roadblocks.
- The Daily Scrum is held at the same time and place to avoid the complexities and this also ensures that team collaboration and performance is optimized.
- The meeting is structured by the development team and it is the internal meeting of the development team.
- The Development Team or team members often meet immediately after the Daily Scrum for detailed discussions, or to adapt, or re-plan, the rest of the Sprint's work.
- The Scrum Master ensures that the Development Team conducts the meeting, but the Development Team is responsible for conducting the Daily Scrum.

Facilitator Notes:

Explain the participants about Daily Scrum.

The Daily Scrum is conducted for the Development Team everyday of the Sprint. It is a 15-minute time-boxed event, and is the key inspect and adapt meeting. During the Daily Scrum, Development Team plans work for the next 24 hours.

Using the Daily Scrum, the Development Team inspects the progress that has happened towards the Sprint. Daily Scrum also optimizes the probability that the Team will achieve the Sprint Goal at the end of a Sprint. The Development Team should plan the ways to achieve the Sprint Goal and to make sure that the expected Increment is built at the end of the Sprint.

The Development Team sets the structure of the Daily Scrum, and there is more than one way of conducting this meeting. Some of the most commonly answered questions during the Daily Scrum are as follows:

- What was done the previous day that helped the Development Team meet the Sprint Goal?
- What will be done today to help the Development Team meet the Sprint Goal?
- Is there any roadblock that prevents the individual or the Development Team from meeting the Sprint Goal?

The Scrum Master has to monitor and make sure that the Development Team keeps the Daily Scrum within the 15-minute time-box. The Daily Scrum is conducted as an internal meeting for the Development Team. If other members are present, it is the responsibility of the Scrum Master that they do not disrupt the flow of the meeting. Daily Scrums are intended to improve communications, eliminate other meetings, identify impediments to development that have to be removed, highlight and promote quick decision-making, and improve the knowledge of the Development Team.

1.20.3 Sprint Review

Following are the key details of Sprint review:

- A Sprint Review is held at the end of each Sprint to inspect the Increment and adapt the Product Backlog if needed.
- The Product Owner reviews the PBIs selected during the Sprint Planning Meeting and explains the ones that are considered done.
- The Product Owner and stakeholders convert their feedback to new PBIs and the Product Owner does the ordering.
- The Sprint Review Meeting is helpful for external stakeholders, including the end users.
- Sprint review is time-boxed to 4 hours for a one-month Sprint and is proportionally reduced for shorter Sprints.

Facilitator Notes:

Explain the participants about Sprint review and the happenings of a sprint review.

Sprint Review is the event that takes place at the end of each Sprint. The purpose of this meeting is to check the Increment and based on the feedback, to adapt the Product backlog, if it's necessary. The tasks done during the Sprint will be discussed by the Scrum Team and the stakeholder. Sprint Review is primarily done for value optimization and this is done based on the tasks done and the changes made to the Product Backlog during the Sprint. Compared to the Daily Scrum, this is an informal meeting to demonstrate the Increment, get feedback in a collaborative fashion.

It is the responsibility of the Scrum Master to ensure that the Sprint Review happens without fail at the end of every Sprint and it is time-boxed.

According to the official Scrum guide, the events that take place during the Sprint Review are as follows:

- Attendees include the Scrum Team and key stakeholders invited by the Product Owner;
- The Product Owner explains what Product Backlog items have been “Done” and what has not been “Done”;
- The Development Team discusses what went well during the Sprint, what problems it ran into, and how those problems were solved;
- The Development Team demonstrates the work that it has “Done” and answers questions about the Increment;
- The Product Owner discusses the Product Backlog as it stands. He or she projects likely target and delivery dates based on progress to date (if needed);
- The entire group collaborates on what to do next, so that the Sprint Review provides valuable input to subsequent Sprint Planning;
- Review of how the marketplace or potential use of the product might have changed what is the most valuable thing to do next; and,
- Review of the timeline, budget, potential capabilities, and marketplace for the next anticipated releases of functionality or capability of the product.

At the end of the Sprint Review, the Product Backlog is generally revised such that it defines the PBIs for the next Sprint. Adjustments can also be made to accommodate the features raised in the feedback and meet the requirements.

1.20.4 Sprint Retrospective

- During the Sprint retrospective, the team inspects itself and creates a plan for improvements that are to be implemented during the next Sprint.
- The Sprint retrospective occurs after the Sprint review and prior to the next Sprint planning.
- This meeting is time-boxed to three hours for a one-month Sprint and is proportionally reduced for shorter Sprints.

The purpose of the Sprint retrospective is to:

- Inspect as to how the last Sprint went in terms of people, relationships, processes and tools
- Identify the important items that went well and the areas for improvement
- Plan for implementing improvements in the Scrum team

Facilitator Notes:

Explain the participants what happens during Sprint retrospective.

The primary intention of the Sprint retrospective is to do a self-inspection and identifying the areas of improvement and planning to implement those improvements in the subsequent Sprints, based on the learning from the current Sprint. This meeting happens after the Sprint review and is time-boxed to three hours for a typical one-month Sprint and is reduced proportionally for shorter Sprints.

It is the responsibility of the Scrum Master to ensure that the meeting happens, it is productive and it solves the intended purpose. Here, the Scrum Master participates as a team member who is accountable for the Scrum process. Improvements for the Scrum team are suggested by the Scrum Master, so that they can be applied during the next Sprint. The focus here is to improve the quality of the product by means of improving the processes or adjusting the definition of 'Done'. Scrum retrospective is not the only time where the team thinks about improvements, but it is a formal opportunity to focus on inspection and adaptation.

What did You Grasp?



1. What is the maximum time-box for a sprint?
A) 1 month
B) 1 week
C) 1 day
D) 15 days
2. Which of the following events is considered as an internal meeting for the development team?
A) **Sprint Planning**
B) Daily Scrum
C) Backlog Grooming
D) Sprint Review

Facilitator Notes:

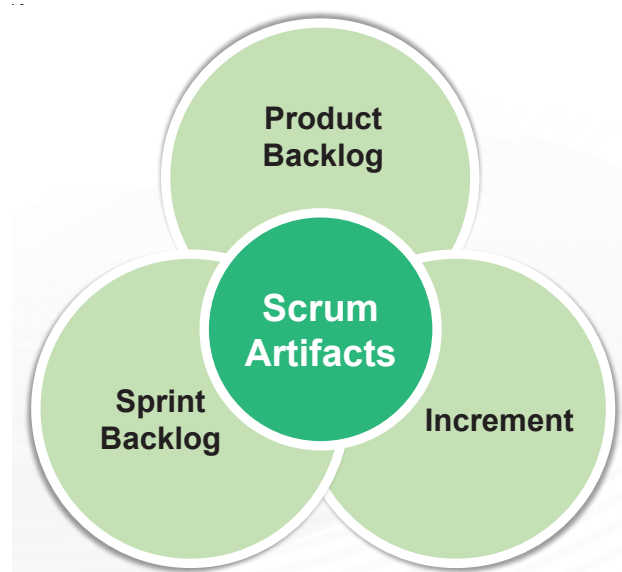
Tell the participants that they will be going through a knowledge check question.

Answers:

1. a. 1 month
2. b. Daily Scrum

1.21 Scrum Artifacts

Scrum defines three artifacts:



Facilitator Notes:

Give the participants an overview of Scrum artifacts.

Scrum defines three major artifacts.

- Product Backlog
- Sprint Backlog
- Increment

We'll look at each of these in detail in the upcoming sections.

1.21.1 Product Backlog

The salient features of the product backlog artifact are as follows:

- Product backlog is a list of the desired functionalities in the order of priority.
- All stakeholders involved have access to the product backlog.
- Items can be added to the product backlog by the development team on discretion of product owner and Scrum Master.
- Product owner continuously reorders the PBIs.
- It undergoes constant refinement by the Scrum team.
- Items at the top of the product backlog should be smaller than the ones at the bottom.
- During product backlog refinement, more details, estimates and the order to the items in the product backlog are added.

Facilitator Notes:

Explain the participants about product backlog and its features.

The product backlog is the list of all the items that has to be there in the product, ordered in terms of priority. Product backlog serves as the one single source of all the requirements for any change to be made to the product. The product owner decides what goes into the product backlog, the availability and the order of the items. In general, a product backlog is an evolving one. During the initial stages only the known and the best understood requirements

go into it. As the product gets developed and new functionalities are added, the priorities may change, hence the product backlog also changes. The product backlog, thus, keeps changing to identify the requirements that make the product appropriate, useful and competitive. As long as the product exists, the product backlog exists. At any point in time, it gives a picture of what needs to be done in the product while value is delivered.

The product backlog is a comprehensive list of features, functions, requirements, enhancements, and fixes needed for the changes to be made to the product in future releases. The attributes of PBIs are as follows: description, order, estimate, and value. There are also test descriptions that will prove its completeness when “Done”. As the product is used, and receives feedback from the market, the product backlog becomes a larger and exhaustive list. Since the requirements keep changing, the product backlog becomes a living artifact. Some of the factors that cause changes in a product backlog are: change in business requirements, market or technology landscape. Most often, multiple Scrum teams work on the same product and the product backlog lists the upcoming work on the product.

Product Backlog Refinement

Backlog refinement is an exercise during which more details, estimates and order to items in the product backlog are added. In this process, the product owner and the development team collaborate on the PBIs and it is an ongoing process. During refinement, the PBIs are revisited and revised. The Scrum team determines the time and the methods for doing the refinement. This refinement exercise generally do not consume more than 10% of the capacity of the development team. The product owner or the team at the product owner’s discretion, can update the PBIs at any time.

PBIs that have a higher order are more clear and detailed than the ones that have a lower order, hence the higher order ones can be estimated more precisely than the lower order ones. PBIs that will go into the upcoming Sprint can be refined, in such a way that they can be ‘Done’ within the Sprint timebox. PBIs that can be done within the timebox of a Sprint are considered ‘Ready’ for selection in a Sprint planning. Through the refining activities the PBIs gain transparency.

The development team owns the responsibility for the estimates, since they are the people who perform the work. The product owner can help the development team to understand and select trade-offs during this estimating process.

1.21.2 Sprint Backlog

Sprint backlog is the set of PBIs selected for the Sprint, along with the plan to realize the Sprint goal.

- Every Sprint backlog consists of PBIs selected by the team and the product owner for the Sprint
- Sprint Backlog also consists of the plan for delivering the Product Increment and realizing the Sprint goal
- In general, Sprint Backlog is a forecast done by the Development Team on what will go into the next Increment and the effort needed to deliver the functionality into a 'Done' Increment
- The Development Team has access to the Sprint backlog
- Sprint backlog is the reference document during the daily Scrum meeting

Facilitator Notes:

Explain the participants about sprint backlog and its features.

Sprint backlog consists of items from the product backlog that are selected for the Sprint, along with the plan for realizing the Sprint goal. It is like a forecast done by the development team on the functionality to be present in the forthcoming Increment and the effort required for delivering the 'Done' Increment. Sprint backlog showcases all the effort required by the development team to attain the Sprint goal. There will at least be one high priority process improvement that was identified in the previous retrospective. During the course of the Sprint the development team constantly modifies the Sprint backlog and it evolves during the Sprint, as the team works and makes progress and the Team learns more about the effort needed to attain the Sprint goal.

If new functionality is required, the development team adds it as an item to the Sprint backlog. As the team works on the project, the estimate of the remaining work is updated accordingly. Sprint progress is generally monitored by tracking the remaining work. If any of the elements are found unnecessary, they are removed from the Sprint backlog. The authority to change the Sprint backlog rests with the development team. Sprint backlog is a live picture of the work that the development team has to do to accomplish the Sprint goal.

1.21.3 Increment

Increment refers to the sum of all the product backlog items completed during a Sprint and the value of the increments of all the previous Sprints.

The salient features of an increment are as follows:

- An increment consists of the product capabilities completed during the Sprints
- An increment should be a usable, releasable state by the end of each Sprint
- An increment is released at the discretion of the product owner
- During the Sprint review Meeting an increment is inspected

Facilitator Notes:

Explain the participants about increment and its features.

An increment refers to the sum of all the product backlog items completed during a Sprint and the value of the increments of all previous Sprints. An increment is said to be 'Done', only if it is in a usable condition and meets the definition of 'Done'. A 'Done' increment is the outcome of every Sprint. Each increment takes the product one step further towards the goal. The authority to release the increments rests with the product owner. The increment should be in a usable form, regardless of the decision of the product owner to release it or not.

We've so far seen about the different aspects of Scrum. We'll now look at the benefits of Scrum.

What did You Grasp?



1. Who is the authority to decide the items that go into the product backlog?
 - A) Scrum Master
 - B) Product Owner
 - C) Development Team
 - D) Customer
2. Which of these artifacts is considered as the reference document during the daily Scrum?
 - A) Product Backlog
 - B) Sprint Backlog
 - C) Increment
 - D) Sprint document

Facilitator Notes:

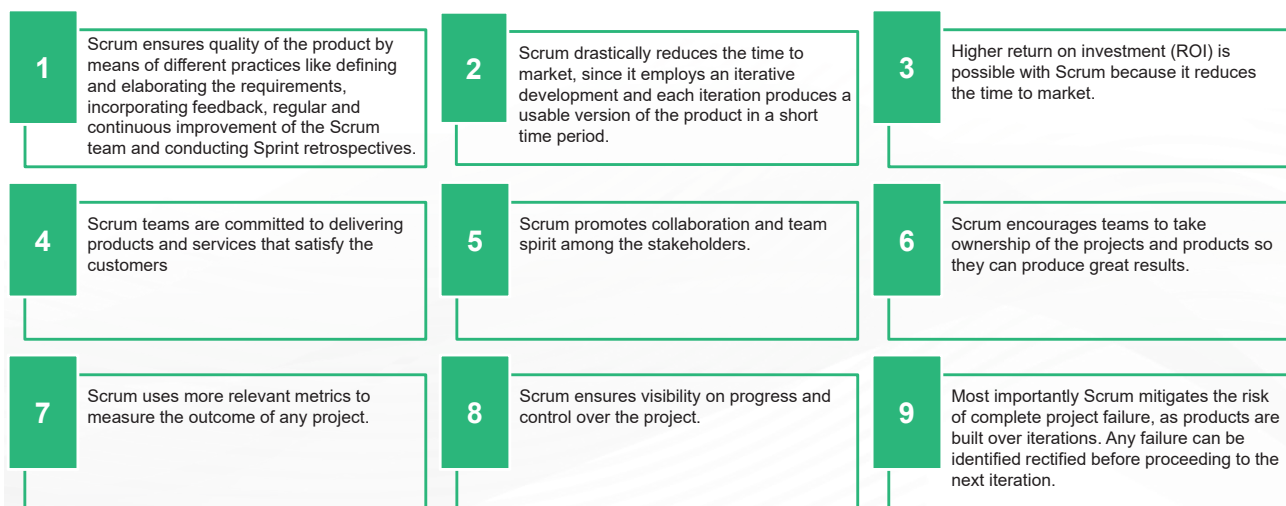
Tell the participants that they will be going through a knowledge check question.

Answers:

1. b. Product Owner
2. b. Sprint Backlog

1.22 Benefits of Scrum

Scrum offers the following benefits over other traditional development methodologies. The key benefits of Scrum are as follows:

**Facilitator Notes:**

Enumerate the benefits of Scrum to the participants.

Go through the pointers in the slide and understand the benefits of Scrum. The meticulous processes that Scrum employs, play a major role in producing the desired product in the desired quality, hence the benefits.

So far, we learnt about the most important and popular agile methodology Scrum. We'll now move on to another popular agile methodology called Extreme Programming (XP).

1.23 Planning and Estimation - An Introduction

- Estimation and planning are not just about determining an appropriate schedule or deadline for any given project.
- Planning for any project should be an iterative and ongoing process to determine what to build and when to deliver it.
- The factors to be considered while planning are the features to be built, available resources and the schedule.
- Planning is a very important step in implementing Agile or any other development method.

Planning is important because it helps in the following:

Reducing risk

Reducing uncertainty

Supporting better decision making

Establishing trust

Conveying information

Facilitator Notes:

Give an overview of planning and why it is needed.

Estimation and planning are the crucial steps to implement Agile in any organization. Agile planning and estimation are not very easy tasks. As customer requirements keep changing, plan and estimates also should change to accommodate the changes. Estimation and planning are more than determining schedule or deadlines. Planning should be an ongoing iterative approach, which is intended towards delivering value. Planning is about determining what to build and the factors that determine this are, features to build, available resources and the schedule. A plan that is created at the start of a project may change in the due course to accommodate the changing requirements, changing priorities and deadlines.

Planning is essential for the implementation of all Agile projects, because it supports:

- **Reducing risk:** Project planning helps the team understand the associated risks. Being aware of the risks and mitigating them early increases the possibility of project success. Teams may decide not to start a project if too much of risk is involved. Some risks can be mitigated by attending to them early. In both these cases planning helps to devise the approach towards reducing or eliminating risks.
- **Reducing uncertainty:** During the course of product development, as new capabilities are introduced into the product, teams gain new knowledge about the product and the technologies used. This newly acquired knowledge should go into refining the product vision. One of the most critical risk factors is that building the wrong product, i.e., the one that the customer does not want. But in most of the cases this risk is ignored. Agile planning helps in reducing and eliminating the risk.
- **Supporting better decision making:** Planning and estimating support the organizations in making better decisions. Any organization has to be aware of the value and cost of the project, to take a decision whether to take it up or not. Estimates also help the organizations make sure that they are working on valuable projects. It also helps in allocating resources for a particular project.
- **Establishing trust:** Trust will build between the customer and organization, only if the organization delivers a high value product with the promised features at frequent intervals. Reliable delivery can be achieved on the basis of reliable estimates. Estimates help the customer decide on priorities and take tradeoff decisions. Using estimates, developers can work at a sustainable pace.
- **Conveying information:** Planning communicates and sets the basic set of expectations. In most of the case plan is distilled down to a single date without the logic that led to those assumptions and expectations. A good plan clearly communicates what will be accomplished during the accepted schedule and the assumptions that led to arriving at the schedule.

We'll cover Agile planning and estimation in this module.

1.23.1 Agile Planning

According to Mike Cohn, the author of Agile Estimating and Planning, "Agile planning balances the effort and investment in planning with the knowledge that we will revise the plan through the course of the project."

- Plan changes only if there is a need that arises based on the learning acquired through the course of the project.
- Plan changes if the customer requirements change, or change in technology or change in priorities.

The most important aspects of Agile planning include:

- It is focused more on the planning than the plan
- It encourages change
- It results in plans that are easily changed
- It's spread throughout the project

Facilitator Notes:

Introduce Agile planning to the participants.

We have seen that Agile planning is about getting answers to questions like what to be built and when it will be delivered, by taking into account the cost and resources involved. Planning also helps managers explore the hidden dependencies on the activities in order to reduce the idle time and optimizing the delivery period. Agile planning also helps in measuring the speed and efficiency of the Agile team.

Mike Cohn, the author of Agile Estimating and Planning, states that “Agile planning balances the effort and investment in planning with the knowledge that we will revise the plan through the course of the project. An Agile plan is one that we are not only willing but anxious to change.” Plan will change only if there is a customer requirement or change in technology or change in priorities. Customers may want to include a new feature or remove an existing feature. Organizations will consider all these factors and their financial impact, before making any alteration to the plan.

An Agile plan is easily changeable and that is why planning becomes more important than the plan itself. Agile plan is one that is easy to change. Changing an Agile plan doesn't mean that the delivery dates will change. The plan can be changed without changing the dates. Agile planning:

- is focused more on the planning than the plan
- encourages change
- results in plans that are easily changed

- is spread throughout the project

We'll now look at the need for Agile planning.

1.24 Need for Agile Planning

Agile planning helps in overcoming the deficiencies associated with traditional planning, which include:

- Focusing more on activities than delivered features
- Ignoring the prioritization
- Not considering the existence of uncertainty
- Considering estimations as commitments

To keep up with the progress of Agile projects and the dynamic environments, the planning has to be Agile as well.

Agile planning factors in the user requirements and the plan is adapted to accommodate the changes, which makes it superior to traditional planning.

Facilitator Notes:

Explain the participants the need for Agile planning.

Traditional planning has deficiencies that can be overcome by Agile planning. These deficiencies include:

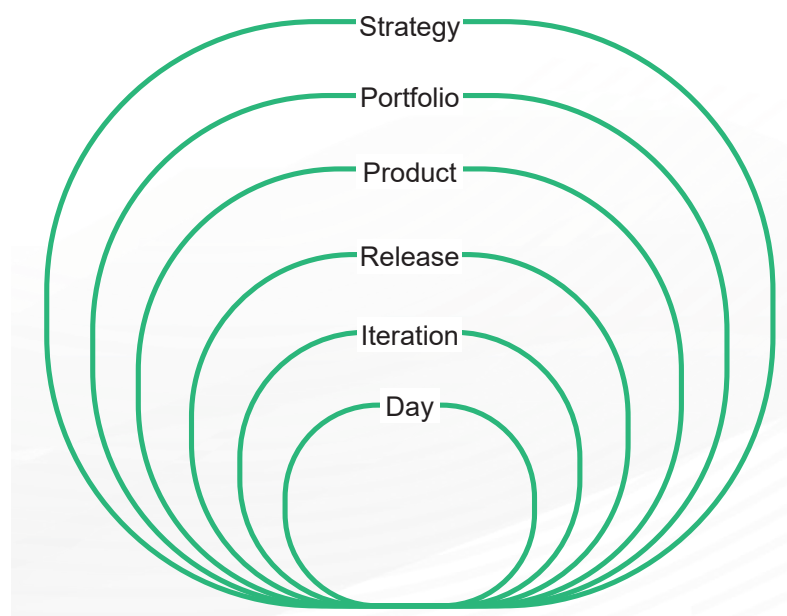
- Focusing more on activities than delivered features
- Ignoring the prioritization
- Not considering the existence of uncertainty
- Considering estimations as commitments

These deficiencies make traditional planning unable to keep up with the pace of Agile projects and they become inappropriate in the dynamic Agile environments. To overcome these issues, planning also needs to become Agile. In Agile planning, changing customer requirements are accommodated and the plan is adapted to suit those changes.

Agile is all about iterative development, and at the start of every iteration, Agile team incorporates the learning acquired during the preceding iteration and adapts accordingly. If the team finds something that will have an impact on the value of the plan, the plan is adjusted. By over or under estimation of the progress rate, the value of the plan might be affected. The value of the features to be included in an iteration might also change during the course of the project. In these cases, the value of the plan can be increased by adding the most desired features into the initial release and the less important features to a future release. Thus, Agile plan is accommodating and adaptable.

1.25 The Agile Planning Onion

The given image illustrates the relationship between the different planning horizons.



Credit: Mike Cohn

Facilitator Notes:

Explain the participants about the different layers of the Agile planning onion.

An Agile planning is more like a timed race. You're given a certain amount of time. And you're expected to reach as far as possible at that given time.

An Agile project will be at risk if its planning extends beyond the planner's horizon and if there's no time for the planner to look at the new horizons and make adjustments accordingly. Most of the Agile teams consider the three innermost levels for planning, i.e., the release, the iteration and the current day. The relationship between these three and the other horizons are illustrated in the figure above.

Release planning: Release planning takes into account the user stories or themes that will be developed for a new release of a product or system. Release planning is intended to determine an appropriate answer to the questions of scope, schedule, and resources for a project. Release planning is usually done at the start of any project, but it doesn't end there. It is updated throughout the project, to make sure that it is in lines with the current expectations about the features that will be included in the release. Updates to the release planning is usually done at the start of every iteration.

Iteration planning: Iteration planning is carried out at the start of each iteration. Based on the accomplishment in the previous iteration, the product owner identifies the tasks that are of high priority that the team should address in the upcoming iteration. Iteration is an even closer horizon compared to a release and thus the components of iteration planning could be smaller. During iteration planning, teams will list down the tasks that are required to transform a feature request to a usable software, i.e., working and tested.

Daily planning: Daily planning is carried out by the teams to organize the work and daily efforts. During the daily planning meeting, teams make, assess and revise their plans. The limit for the daily planning is the current day, considering the fact that they will meet again the next day. Teams plan on coordinating the individual activities that help in task completion.

By means of planning across the above three horizons, teams can have their focus on the visible and important components of the product.

The fourth level, product planning involves a product owner looking further ahead than the immediate release and planning for the evolution of the released product or system. In portfolio planning, products are selected according to the vision established through an organization's strategic planning.

A slightly updated version of the Agile planning levels and the description of those levels is given in detail in the upcoming slides.

What did You Grasp?



1. State True or False.
In Agile planning, the plans cannot be changed easily.
A) True
B) False
2. Which among the following is the most visible horizon in the Agile planning onion?
A) Day
B) Iteration
C) Release
D) Product

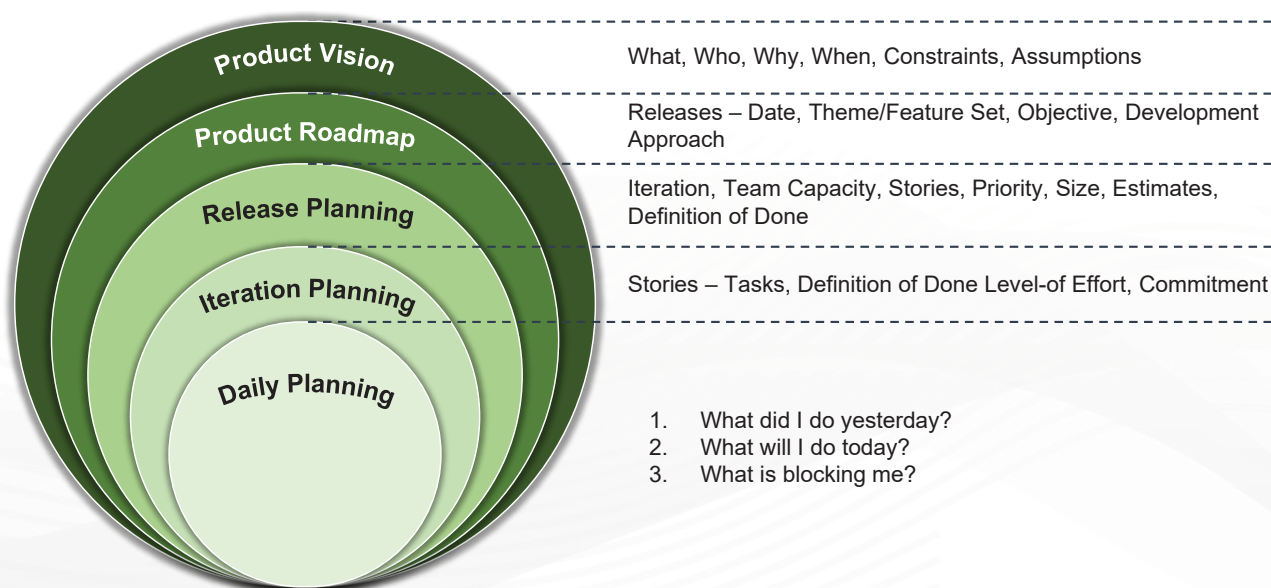
Facilitator Notes:

Tell the participants that they will be going through a knowledge check question.

Answers:

1. b. False
2. a. Day

1.26 Levels of Agile Planning



Facilitator Notes:

Explain the participants about the different levels of Agile planning and the questions answered and the tasks carried out in each level.

The above illustration shows the five different levels of Agile planning. From the picture you can also understand the tasks carried out in each level and the questions answered during those levels. The five levels of Agile planning are as follows:






- Product vision
- Product roadmap
- Release planning
- Iteration planning
- Daily planning

We'll look at each of these in detail in the upcoming sections.

1.26.1 Product Vision**Following are the key details of product vision:**

- The outermost level of the planning horizon and it concentrates on the future of the product.
- The macro image of the product, i.e., how it will look at the end of the project, is defined at this level.
- Product vision is set primarily by the product owner, with a little support from the project manager.
- Product vision creation step is to ensure that the strategies are aligned properly and the team spends the time and effort only on creating and delivering the valuable product.
- The vision statement also tells us how the product supports the organization's strategies. The picture shows a product vision template.

THE PRODUCT VISION BOARD

 VISION			
 TARGET GROUP	 NEEDS	 PRODUCT	 BUSINESS GOALS

Facilitator Notes:

Explain the participants about product vision planning.

Product Vision is the broadest picture of how the product will look at the end of the project. The product owner establishes the priority and the effort involved in achieving the goal.

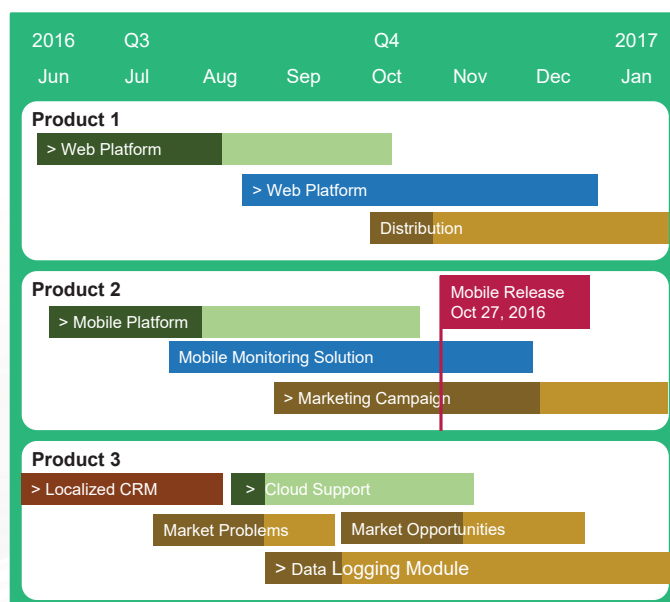
The product vision describes the state of the product in six months or more in the future. Further activities on planning will give more details about the vision and there may even be diversions from the vision, since the perspective might change in terms of market, product and the effort required to achieve the vision.

Two common ways of vision exercise are to create an elevator statement or a product box. Both these exercises are intended to create that statement which describes the future of the product in terms of product features, target customers, and key differentiators from other products.

1.26.2 Product Roadmap

Following are the key details of product roadmap:

- Roadmap is the plan that describes the way that the product is likely to grow.
- Agile teams focus on a goal-oriented roadmap, as it is important for the team to know about the everyday work.
- Product roadmap also helps in estimating the budget for developing and delivering high value product as per the promised schedule.
- Agile environment is prone to changes because of the changing requirements from the customers, hence creating a roadmap is often a challenging task.
- Product roadmap should focus on goals, benefits, objectives, acquiring customers and increasing their engagement. The picture shows a sample product roadmap.

**Facilitator Notes:**

Explain the activities carried out during product roadmap planning.

Customers demand for changes at more frequent intervals. Time to market in these days is measured in weeks or months and not in years, which was the case in traditional projects. This makes the product owner devise a roadmap or path towards the final product. A product roadmap is created and communicated to the fellow team members.

A product roadmap is created primarily by the product owner in a single meeting or a series of meetings. The roadmap will contain the details of the dates, content, and objectives of the foreseen releases. The product backlog is also created, which includes the list of desired features and the priorities.

1.26.3 Release Planning

Following are the key details of release planning:

- Release is a set of increments that are released to the customer at frequent intervals.
- The release plan helps to estimate the amount of work that will be delivered by the team at the scheduled deadline.
- Release planning is a collaborative effort of the Scrum Master and the product owner, Agile team and the stakeholders.
- Release planning is focused on goals and benefits, by taking into account, the dependencies and uncertainties.

Facilitator Notes:

Explain the participants about release planning.

During the release planning, a set of activities is grouped into tasks and assigned to the teams. A release is the set of product increments that are released to the customer. Some of the important characteristics of a release are as follows:

- Releases will have the date, theme and a set of features
- In release planning the scope, and not date or quality, is defined, hence a product backlog is required as the base of the planning event

- Teams should work in the same rhythm for better management of dependencies
- There are fixed release dates across all team of the program with a typical interval of two to four months

In general, a release planning session takes place over a day, in case of large teams, it may take two days. This session is attended by all the team members, including product owner, delivery team, and stakeholders. Release planning should be as collaborative and interactive as possible.

A typical agenda for the release planning meeting could be:

- Introduction and goal setting
- Explanation of product vision
- Time-boxes for the releases and iterations
- Capacity calculation by the delivery team
- Agreement of deliverables (when is a feature 'Done')
- Moving features from the backlog into the iterations within the release by the individual teams on the basis of priority
- Dependency determination, by means of analysing the individual planning results
- Estimation of workload per iteration, in accordance with the available capacity
- Review of discovered risks and issues
- Retrospective of the session

1.26.4 Iteration Planning

Following are the key details of iteration planning:

- Iteration planning is generally a subset of the release plan stories that will be done in the next iteration or a sprint.
- During iteration planning, the volume of the backlog items that a team can deliver during the next iteration, are determined.
- The team commits to deliver the features according to their velocity and the schedule.
- Iteration plan is about breaking down the features into multiple development tasks and estimating them.
- Use cases are also created to identify the stories that will be developed and to break them into specific tasks and acceptance criteria.

Facilitator Notes:

Explain the participants about iteration planning.

Individual iterations within a release are planned at this level. This exercise is done to add more detail and increasing the accuracy. In this phase, features are broken down into tasks. Teams commit to the features that will be delivered during the iteration, with a higher degree of certainty.

An iteration planning has a similar structure to that of a release planning session, the primary difference being the planning horizon. Teams work individually to produce their iteration plans, the synchronization between them help them in detecting and resolving dependencies.

The core of the activities of iteration planning is carried out on a team-by-team basis:

- Actual capacity and the amount of work that can be done by the teams is determined
- Teams break down the features into tasks
- Task sizes are estimated with the task size being half-day to two days
- The definition of 'Done' is taken into consideration, a feature is completely tested and accepted by the product owner
- The results of the individual teams are inspected in a combined session to determine the hidden dependencies

1.26.5 Daily Scrum/Standup

Following are the key details of daily scrum/standup:

- Daily scrum is done to coordinate the work and synchronize the effort.
- Teams make, assess and revise their plans during this meeting.
- Daily scrum is focused on completing the top priority features. This meeting is focused on the individual's accomplishments, plan for the day and the issues faced by them and ways to resolve them.
- For the unresolved issues, 'Parking Lot' can be used. Anything out of the scope of the daily planning will be placed in the parking lot and dealt with later.

Facilitator Notes:

Explain the participants about daily planning.

The daily standup meeting, called Daily Scrum is done at the start of every day to set the context of the current day. This event is time-boxed and the discussion is kept brisk and relevant. The questions that a development team answer during the Daily Scrum are as follows:

- What was done yesterday?
- What will be done today?
- Are there any impediments on the way?

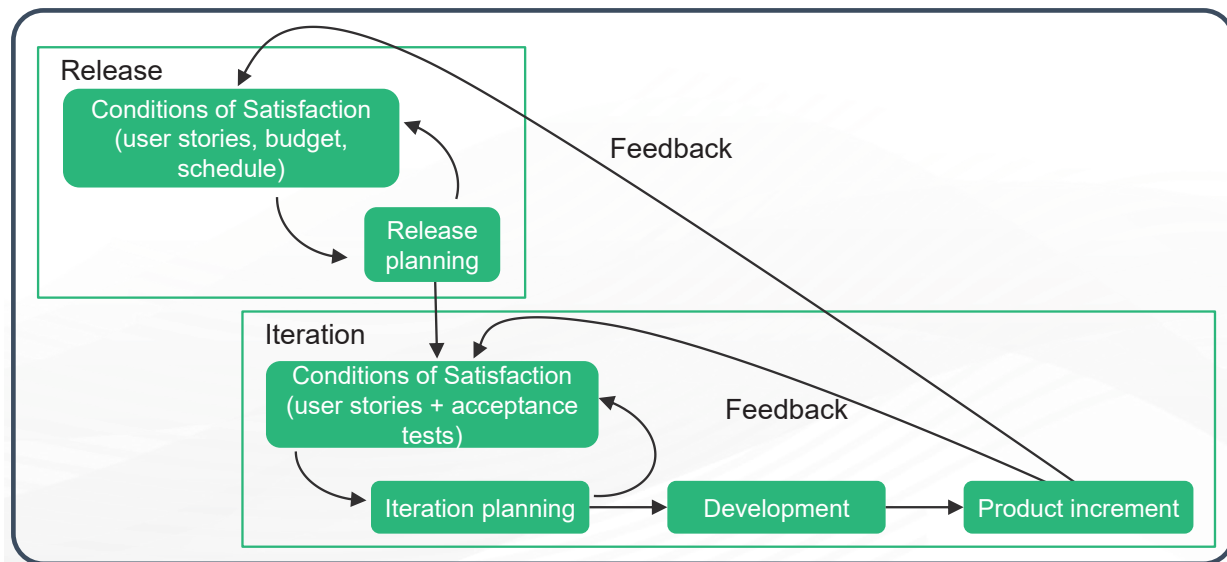
Team members explain the lessons learned from the previous day's session. Issues are raised and addressed.

Organizations use various processes to manage and coordinate projects and associated tasks within teams. Stand-up meetings are one such method widely employed by organizations to track and manage the progress of their internal projects and responsible teams.

Ideally, representatives from teams gather to report the status, plans and changes required to current projects. All teams follow an identical format of reporting. Different team representatives meet on a regular basis, i.e, once a week to address the questions. Stand-up meetings are also conducted within each team to answer the same set of questions. The frequency of the team meetings occurs based on the timelines and project delivery.

1.27 Conditions of Satisfaction

The picture shows how the conditions of satisfaction drive both release and iteration planning.



Facilitator Notes:

Explain the participants about the conditions of satisfaction.


Mike Cohn defined conditions of satisfaction as “the criteria that will be used to gauge the success of the project.”

At the commencement of release planning, the team and product owner collaboratively explore the product owner’s conditions of satisfaction. Product owner’s conditions of satisfaction include the factors such as scope, schedule, budget, and quality, though quality is non-negotiable for Agile teams. The product owner and the team will collaboratively look for meeting all the conditions of satisfaction. In the event of the team not being able to meet any of the conditions, the conditions of satisfaction must change. Because of this reason, release planning and exploration of the product owner’s conditions of satisfaction are highly iterative, which is shown in the figure above. Similarly, for the iteration planning, the product owner and the team collaborate to find out the ways for meeting the conditions of satisfaction.

Feedback loops from the new product increment would go back into the conditions of satisfaction at the start of both the release and iteration.

We'll now move on to Agile estimation. The first concept in Agile estimation that we'll look at, is Estimating the size

What did You Grasp?



1. Who is responsible for devising the product roadmap?
A) Developer
B) Product Owner
C) Scrum Master
D) Customer
2. State True or False
Conditions of satisfaction are checked only at the iteration level.
A) True
B) False

Facilitator Notes:

Tell the participants that they will be going through a knowledge check question.

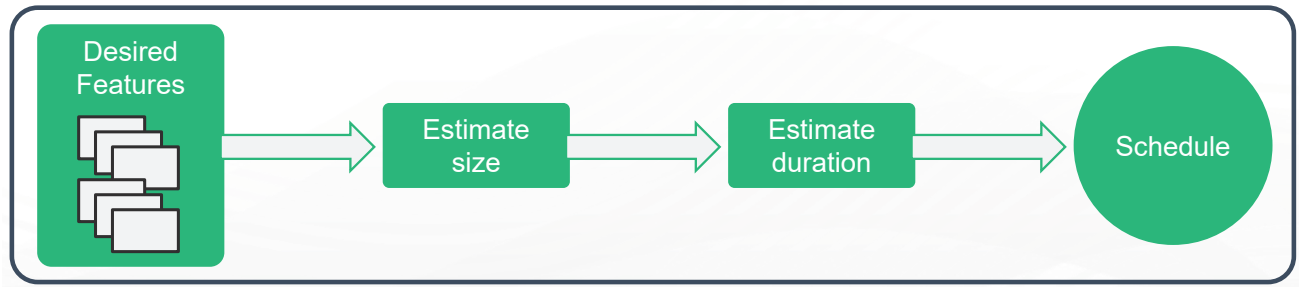
Answers:

1. b. Product Owner
2. b. False

1.28 Estimating the Size - Story Points

Estimate is defined as 'the quantified evaluation of the effort necessary to carry out a given development task.' Estimation is done in terms of size or duration.

- Estimating the size helps in the estimation of duration, which is shown in the figure given below.



- Story points are a unit of measure for expressing the overall size of a user story, feature, or any other piece of work. A point value is assigned to each item while estimating with story points.
- Story points are relative and not absolute. A story point with a value two should be twice as much the story with the value one.
- The number of story points associated with a story represents the overall size of the story.

Facilitator Notes:

Explain the participants about estimating with story points.

Mike Cohn defines story points are a unit of measure for expressing the overall size of a user story, feature, or other piece of work. Point values are assigned to each item when estimation is done with story points. Story point values are relative ones. A story that is assigned a value two will be as twice as a story with value one. The number of story points associated with a story represents the overall size of the story. There is no strict formula for defining the story size. Story size is normally defined on the basis of the amount of effort involved in developing the feature, the complexity of developing it, the risk inherent in it, and so on.

Typically, there are two ways to get started with story points.

1. Select a story that is expected to be smaller, which is estimated at a value of 1 story point.
2. Select a medium-sized story and assign it a number somewhere in the middle of the range that is expected to be used.

Once the first story is estimated the remaining stories can be estimated by comparing them with the first story or the ones that have been estimated already. The first story will serve as the base or reference story for the second story. This way relative sizing is done using story points.

1.29 Velocity

Following are the key details of velocity:

- Velocity is defined as a measure of a team's rate of progress, which is important to understand the story points.
- Velocity is by adding the number of story points assigned to each user story that the team completed during the iteration.
- For example, if a team completes three stories that are valued at three story points each, the velocity of the team would be nine.
- Based on the velocity of the team during the previous iteration, the estimate for the next iteration can be estimated.
- If the story point estimates of all the features are summed up, the total size estimate for the project can be done.

Facilitator Notes:

Explain the participants about velocity and how it helps understand the estimating story points for a project.

Velocity is defined as the rate of the team's progress. Velocity is calculated by adding up the story points assigned to each user story that the team completed during the iteration. For example, if a team completes three stories that are valued at three story points each, the velocity of the team would be nine.

We'll see how we can use velocity to estimate the size. If a team completed ten story points of work last iteration, we can assume that they will complete ten story points this iteration. Since story points are estimates of the relative size, this calculation will hold true if the team completes two five-point user stories or five two-point user stories.

The total size of any project is estimated by adding up the story points of all the desired features that make up a project. If the team's velocity is known, we can derive the number of

iterations by dividing the size by velocity. This duration can then be converted into a schedule by mapping it onto a calendar. An example of this has been given by Mike Cohn.

“For example, suppose all of the user stories are estimated and the sum of those estimates is 100 story points. Based on past experience, we know the team’s velocity to be 11 story points per two-week iteration. Since we can estimate that the project needs 9.1 iterations. We can either round that up to 10 iterations or find one point to remove so that it becomes 9 iterations. Let’s assume we go with a conservative approach and call it 10 iterations. Since each iteration is two weeks, our estimate of duration is twenty weeks. We can count forward twenty weeks on the calendar and that becomes our schedule.”

1.30 Estimating in Ideal Days

In Agile development, user stories or other tasks are estimated in ideal days.

The assumptions behind ideal days estimation are as follows:

- The story being estimated is the only thing that the developer works on
- Everything the developer needs will be on hand when they start
- There will be no interruptions

While estimating the ideal days required to develop, test and accept a user story, it is not necessary to take into account, the overhead of the environment in which the team works.

If this organizational overhead is ignored, ideal days can be thought of as an estimate similar to story points.

Facilitator Notes:

Explain how the estimate is done as ideal days.

On a software project, user stories or other tasks are estimated in ideal days, based on the following assumptions:

- The story being estimated is the only thing that the developer works on
- Everything the developer needs will be on hand when they start
- There will be no interruptions

Estimating in elapsed days instead of ideal days, requires us to consider all of the interruptions that might occur while working on the particular story. If the estimate is done in ideal days, only the amount of time the story will take is only considered. In this way, ideal days are an estimate of size, although less strictly so than story points.

When organizational overhead is ignored, ideal days can be thought of as another estimate of size, similar to story points. Then, an estimate of size expressed as a number of ideal days can be converted into an estimate of duration using the velocity in exactly the same way as with story points.

When estimating in ideal days it is best to associate a single estimate with each user story. If a user story will take four programmer days, two tester days, and three product owner days, it is better to sum those and tell the story as a whole will take nine ideal days.

What did You Grasp?



1. What is the velocity of the team that completes five user stories that have 4 story points each?
A) 10
B) 15
C) 20
D) 30
2. State True or False.
While estimating in ideal days, it is assumed that the developer works only on the story being estimated.
A) True
B) False

Facilitator Notes:

Tell the participants that they will be going through a knowledge check question.

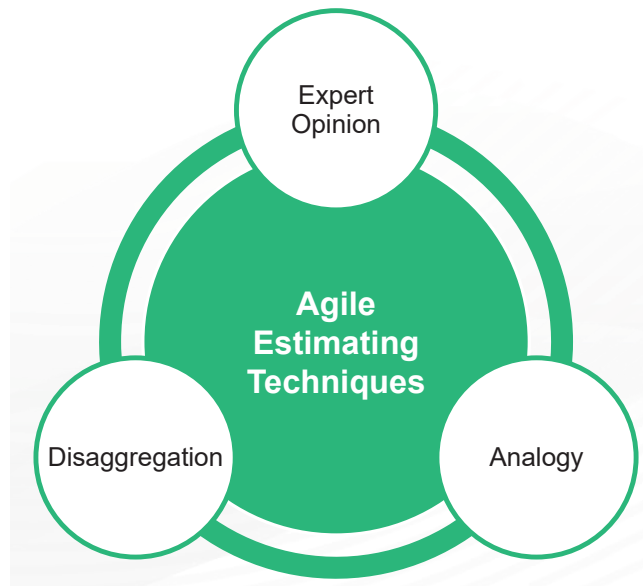
Answers:

1. c. 20

2. a. True

1.31 Estimating Techniques

The three most common techniques for estimating are represented in the figure below:



Facilitator Notes:

Explain the participants about the three techniques for estimating.

Three techniques are commonly used for estimating.

- **Expert Opinion:** In this approach, expert opinion on how long some task will take or how big something will be, is sought out. The expert will provide an estimate based on gut feel or intuition or past experience. This approach is found to be less useful in Agile projects, since Agile estimating is done based on user stories or other important functionalities. Developing a functionality requires varied skill sets and Agile teams are generally cross-functional. It is difficult to find experts who can assess the effort across multiple disciplines. This approach might be suitable for a traditional project, because each task is performed by one person in traditional projects. One benefit of getting expert opinion is that it is a less time consuming option.
- **Analogy:** In this approach, the story being estimated is compared with one or more stories. Then a relative sizing is applied and estimate is provided. A story twice as big as another story will be given an estimate twice as large. In this approach, there is no single base or universal reference. Stories for which estimates are already available are used as benchmarks estimating other stories. This process is also referred to as triangulation.
- **Disaggregation:** Disaggregation refers to splitting down a user story into smaller, easy to estimate pieces. Projects with multiple smaller stories are easier to estimate than the ones with the single

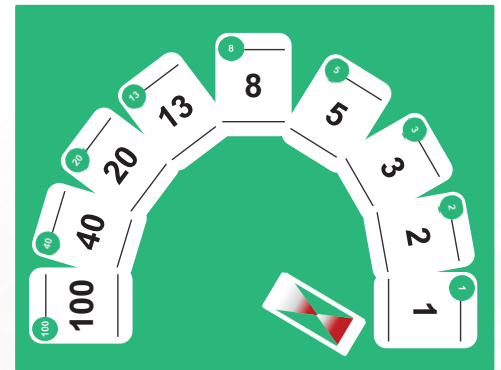
large story. Disaggregation will lead to problems if we go beyond a limit. If we disaggregate too far, the chances of forgetting tasks also increase.

In the upcoming slides, we'll look at two commonly used Agile estimating techniques.

1.32 Planning Poker

Following are the key details of the Planning Poker game:

- Planning poker is one of the most popular gross level estimating technique, which uses Fibonacci sequence (adding previous two numbers to get the next number in the sequence) to assign point value to a story or a feature.
- Fibonacci sequence has been modified to get the following sequence for Agile estimation purposes: 0, 1, 2, 3, 5, 8, 13, 20, 40, 100.
- The numbers in the given sequence are represented in a set of playing cards as shown in the picture.
- Team members play the Planning Poker game and assign a point value to each item. The process is explained below in the notes.
- Fibonacci sequence is used as these numbers to represent relative size and this method also provide the correct level of detail for smaller and better understood projects.



Facilitator Notes:

Explain the participants about the Planning Poker game, which is used for estimating Agile.

Planning Poker is a game-based Agile estimating technique, based on the Fibonacci sequence. In the Fibonacci sequence, the previous two numbers are added to get the next number in the sequence (0, 1, 1, 2, 3, 5, 8,). For Agile estimating the sequence has been modified to get the following sequence: 0, 1, 2, 3, 5, 8, 13, 20, 40, 100. These numbers are printed on playing cards. Team members play the Planning Poker game to assign point value to each story or item. The steps in the game are as follows:

- Each member in the team gets a deck of cards that read the sequence 0, 1, 2, 3, 5, 8, 13, 20, 40, 100. The team includes programmers, testers, database engineers, analysts, user interaction designers and so forth. Ideal team size is 10; the number of team members increases, it is better to split into two teams. Each team will estimate independently. Product Owner participates, but doesn't estimate.
- The moderator reads out the story or the item to be estimated. Usually, the product owner or the analyst will serve the role of the moderator. Product owner answers the questions of the estimators.
- Each member of the team privately selects a card that represents his/her estimate.
- When all the team members are ready with their estimates, all the cards are revealed at the same time.
- If all the team members selected the card with the same number, then that number will be assigned as the point value of the item. If the cards are not the same team then they discusses the item with the emphasis placed on values.
- The member who selected the lowest and the member who selected the highest value should explain the logic behind the selection.
- Selection happens until the numbers converge.
- For longer conversations, teams may use a 2-minute timer to timebox the session.
- The process is repeated for each story or each item.

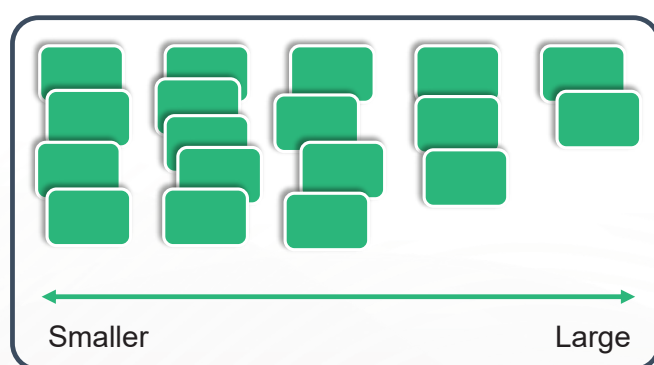
Fibonacci numbers are used, as they represent relative size and the estimate can be done quickly. This sequence also provides right estimates for smaller and better understood items. Teams have only less number of choices for assigning the point values and the estimate are done at a fast pace.

Items with a point value of 20 or higher may be considered as less-understood items. Items with point estimates between 1-13 can generally be completed in a single iteration of 1 - 4 weeks. One important thing to note here is that the points may have different meanings for different teams and team velocity is not the right way to compare productivity across the teams.

1.33 Affinity Grouping

Following are the key details of affinity grouping:

- Affinity grouping is a faster way to Agile estimating, especially when the number of items to estimate is very large.
- Stories or items that are like-sized are grouped together that results in affinity groups as shown in the picture.
- The first item is read out by the presenter to the team members and placed on the wall. The second item is then read out and compared with the size of the first item. If it is of smaller size, it goes to the left of the first one and if larger, goes to the right.
- All the items are thus read out, assessed and placed on the wall. The team then reviews the grouping and any inappropriate placement is modified.
- Once the affinity grouping is complete, point values are then assigned.



Facilitator Notes:

Explain the participants about affinity grouping.

Affinity grouping is a faster way to estimate, especially if the number of items to be estimated is large. Team members analyse and group the items or stories and group similar-sized items to get a structure like the one shown in the picture. This method is simple and fast. The method is explained as follows:

- The first item is read out by the presenter to the team members and placed on the wall.

- The second item is then read out and the team discusses and finds out if it is larger or smaller than the first one. It is then placed on the wall based on the team's response. If larger it goes to the right of the first one, if smaller it will go to the left.
- The third item is then read out and compared with the first two. Based on the size it is placed on the wall.
- The team then gets the control to finish the remainder of the items.
- The team can continue the same way of discussing and placing the items on the wall. Another faster alternative is to have each team member, select one item and place it on the wall as per their discretion. This is done with all the team members, until all the items in the project are assessed and placed on the wall.

This way, several items can be estimated in a short period. Once all the items are placed on the wall, teams then review the grouping. If any team member believes that any item is placed in the wrong group, it can be moved to the appropriate group after discussion.

Once this grouping is complete, point values are assigned to the items, based on the sequence discussed in the previous section.

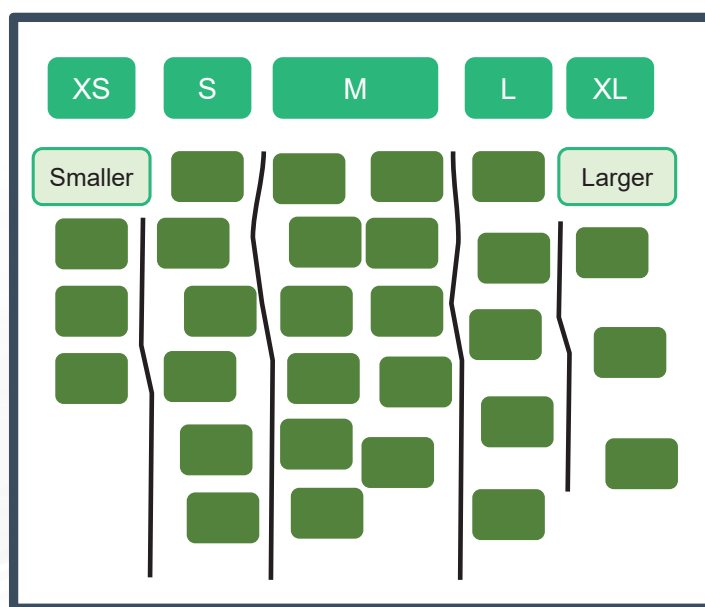
Thus, this section explains the estimation procedure and techniques involved. We'll now look into implementation of Agile in industry projects.

1.34 T-Shirt Sizes

The salient features of T-shirt sizes are as follows:

- T-Shirt sizes is a way of estimating relative sizes of features. Five different sizes Extra Small (XS), Small (S), Medium (M), Large (L), Extra Large (XL) are used for estimating the size of features.
- Using T-shirt sizes is an informal and fast way of assessing the size of backlog items.
- The picture shows how affinity grouping is done using T-shirt sizes.

- After mutual discussion and agreement of the team, a relative size is first decided, which will be medium (M) most often.
- Numbers are then assigned to the items, according to the relative size that is assigned to the medium size.



Facilitator Notes:

Explain the participants about T-shirt sizes for estimating the size of agile projects.

T-Shirt sizes can be used for making high level estimates. T-shirt sizes can be used for estimating a large backlog of relative large items. This method will be useful especially when we have several concurrent scrum teams working on the same product. Items are estimated into T-shirt sizes: XS, S, M, L, XL, as given in the slide above. All the estimators will assign their own size to their items and a final consensus is reached to get a final estimate.

One of the disadvantages of this method is that the sizing may not be uniform. What seems to be L for someone, may be XL for someone else.

What did You Grasp?



1. What is the ideal number of points can be completed in a single iteration?
 A) 0-13
 B) 15-18
 C) 20-25
 D) 25-30
2. State True or False.
 Affinity grouping is suitable only for projects with minimum number of items.
 A) True
 B) False

Facilitator Notes:

Tell the participants that they will be going through a knowledge check question.

Answers:

1. a. 0-13
2. b. False

1.35 Agile Implementation in Industry Projects

Implementation of Agile across organizations can be done by several approaches. Some of these for Developers and Management include:

Developers	Upper management
<ul style="list-style-type: none"> → Resistance → Micromanagement → Transition from traditional to Agile processes → Distributed development → Top talent is required → Enthusiastic teams → Testers 	<ul style="list-style-type: none"> → Customer Commitments → Tracking Progress → Impact on other groups → Project completion

Facilitator Notes:

Explain the participants about the approaches in implementing Agile in industry projects.

So far, we have seen about Agile planning and estimation, which are critical steps in implementing any Agile project. This section lists down the various approaches through which Agile can be implemented in an organization.

1.36 Soft Skills in Agile

Soft skills refer to the interpersonal skills and the ability to interact effectively and harmoniously with other people.

Agile teams and the project leaders need to have soft skills, a few of them are as follows:

**Facilitator Notes:**

List down and explain the most important soft skills that Agile project leaders should have.

Agile soft skills refer to the collaboration and leadership skills the Agile team members and leaders should possess. Some of the important ones are described below:

- **Collaboration:** Collaboration refers to working together with others on a common goal that can otherwise not be achieved alone or by a single group. This may involve planning together, resource pooling and evaluating together. Customer collaboration is a good way to measure project success, by taking into account their feedback throughout the life cycle of a project.
- **Self-leadership:** Agile leaders should lead themselves first in order to lead others. Agile leaders should avoid leaking emotions or projecting on others. They should practice what they preach and lead by example. They should have the clarity of what they want to do.
- **Adaptive leadership:** The main aim of this is to adapt organizations to internal and external pressures for change. Adaptive leadership helps imbibe a positive work culture in an organization. Adaptive leadership is two-dimensional, i.e., being Agile and doing Agile. The four key levers for change that every Agile leader should start with, is:
 - Do Less
 - Speed-to-Value
 - Quality
 - Engage/Inspire

The four principles that adaptive leadership applies for encouraging the engagement of followers in helping the organization to adapt its environment are as follows:

- Understanding the purpose of the organization
- Utilization of people's skills and expertise in helping with adaptation
- Tolerating ambiguity
- Freedom to act
- **Negotiation:** Negotiation is a process by which two or more parties who are in conflict coming together to find a mutually acceptable resolution for the conflict. Successful negotiation needs to have the following qualities:
 - Separate people from the problem
 - Focus on interests, not positions
 - Invent options for mutual gain
 - Use objective criteria
 - Five level of conflicts

- **Agile servant leadership:** Servant leadership refers to both a leadership philosophy and set of leadership practices. A servant leader has the qualities of deep listening, self-awareness and commitment to others. Agile leaders should not tell the teams what to do. They should help the team in the process of self-organization and expediting their progress. An example is that a Scrum master should ensure that the daily standup happens and is conducted in a proper way and estimate the team's work.

Robert Greenleaf, the founder of the modern Servant Leadership has listed the principles of servant leadership which include listening, empathy, awareness, persuasion, conceptualization, foresight and stewardship.

- **Agile coaching:** Agile coach helps the team grow strong in applying Agile practice to their work. Agile coaches should have the following qualities:
 - Coach the team through change
 - Accept the team's ideas above their own
 - Navigate conflict
 - Integrate paths towards high performance

What did You Grasp?



1. Which of the following could not be a soft skill of an Agile leader?

- A) Dictatorship
- B) Self-leadership
- C) Coaching
- D) Negotiation

Facilitator Notes:

Tell the participants that they will be going through a knowledge check question.

Answer:

1. a. Dictatorship

1.37 Lean Thinking

Lean methodology derives itself from Lean Principles in Manufacturing where more is achieved from less. Lean thinking is governed by core principles such as:



Facilitator Notes:

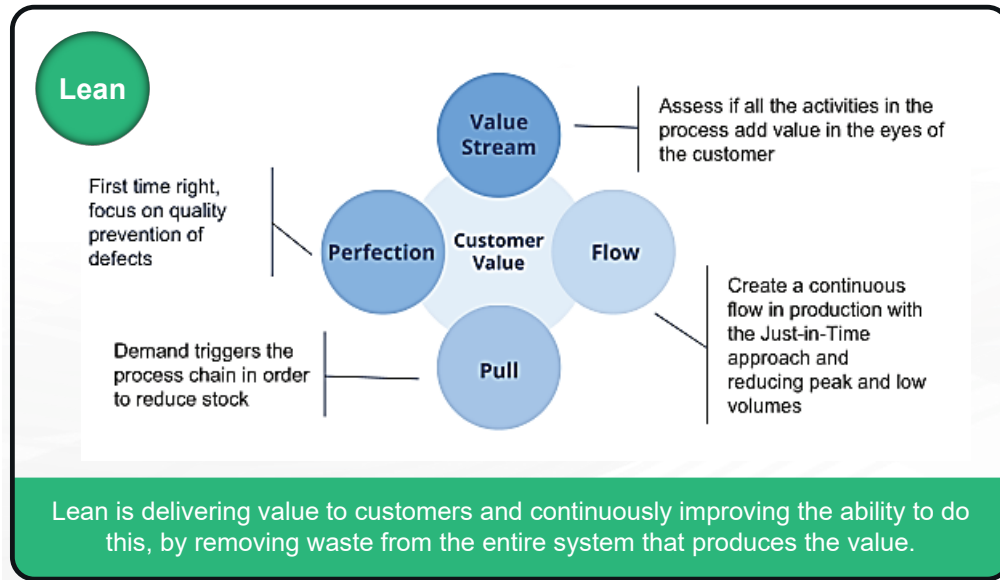
Explain to participants the principles behind lean thinking and how it can be applied to software development methodology.

Lean methodology was derived from lean principles that were applied in lean practices of manufacturing. The crux of the idea is to see how waste can be eliminated and how software development can be streamlined. In lean, waste is explain in a very broad sense and it refers to any activity that be eliminated to streamline the process.

Lean methodology in software is driven by the above mentioned principles of elimination of waste, amplified learning, an empowered team, fast delivery of features or functionality. Considering the high demand for products should be decided as late as possible in such a way that there is no rework. Most importantly, analyzing every single component of the process and having a holistic view is important in lean.

1.38 Lean Methodology: Customer is King

Lean places supreme importance on creating value for the customer.




Facilitator Notes:

Explain to participant the principle of placing the customer before everything else.

Lean Methodology in software development forms the fundamental core on which many other methodologies such as Agile, MVP and DevOps came into existence. Lean methodology assesses the entire value system and development lifecycle for pitfalls/loopholes that causes waste in terms of time, man effort or quality. By studying the value and its components continuously, we focus on continuous improvement of the process and quick delivery of product.

What did You Grasp?



1. Which of the following options is not a LEAN principle?

- A) Elimination of fees
- B) Slow delivery of features
- C) Late decision making
- D) Holistic approach

Facilitator Notes:

Correct Answer

1. B. Slow delivery of features

In a nutshell, we learnt:



1. Agile methodology
2. Software, History of Software Engineering and Software Development Methodologies
3. Traditional Software Development Models
4. Waterfall Model, Classical Waterfall Model
5. Traditional IT Organizations
6. Developers vs IT Operations Conflict
7. Birth of Agile, Four Values of the Agile Manifesto
8. Scrum, Scrum Theory, Scrum Values, Scrum Roles, Scrum Master
Scrum Sprints, Benefits of Scrum
9. Planning and Estimation, Agile Planning, Levels of Agile Planning
10. Conditions of Satisfaction, Velocity
11. Estimating Techniques
12. Soft Skills in Agile
13. Kanban Model