UPES

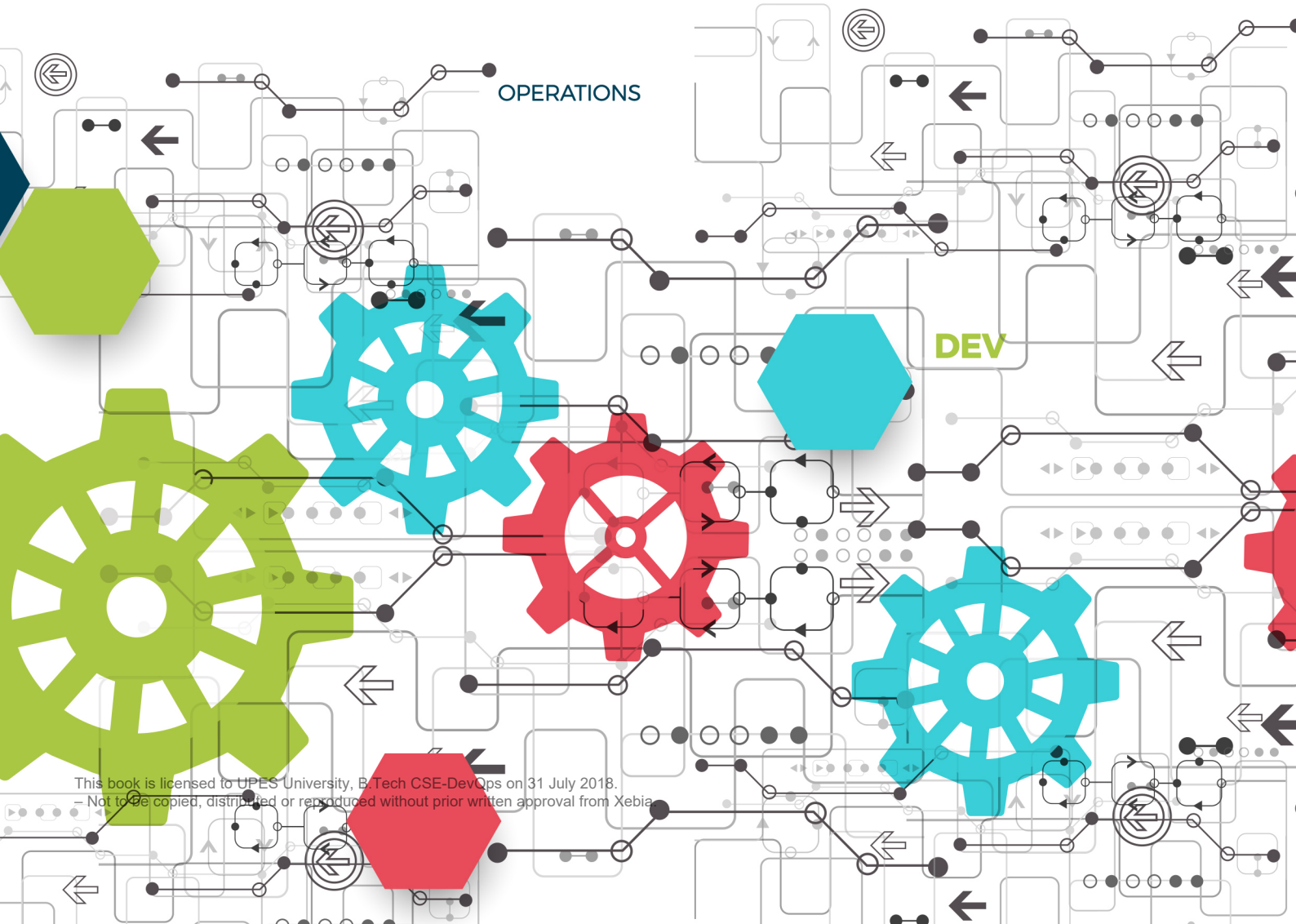
Xebia

B.Tech Computer Science and Engineering in DevOps

DEVOPS OVERVIEW

Semester **01** | **Student Handbook**

Release 1.0.0



Copyright & Disclaimer

B. TECH CSE with Specialization in DevOps

Version 1.0.0

Acknowledgements

We would like to sincerely thank the experts who have contributed to and shaped B. TECH CSE with Specialization in DevOps. Version 1.0.0

SME

Rajagopalan Varadan
A tech enthusiast who loves learning and working with cutting-edge technologies like DevOps, Big Data, Data science, Machine Learning, AWS & Open stack

Course Reviewers.

Aditya Kalia | Xebia
Maneet Kaur | Xebia
Sandeep Singh Rawat | Xebia
Abhishek Srivastava | Xebia
Rohit Sharma | Xebia

Review Board Members.

Anand Sahay | Xebia

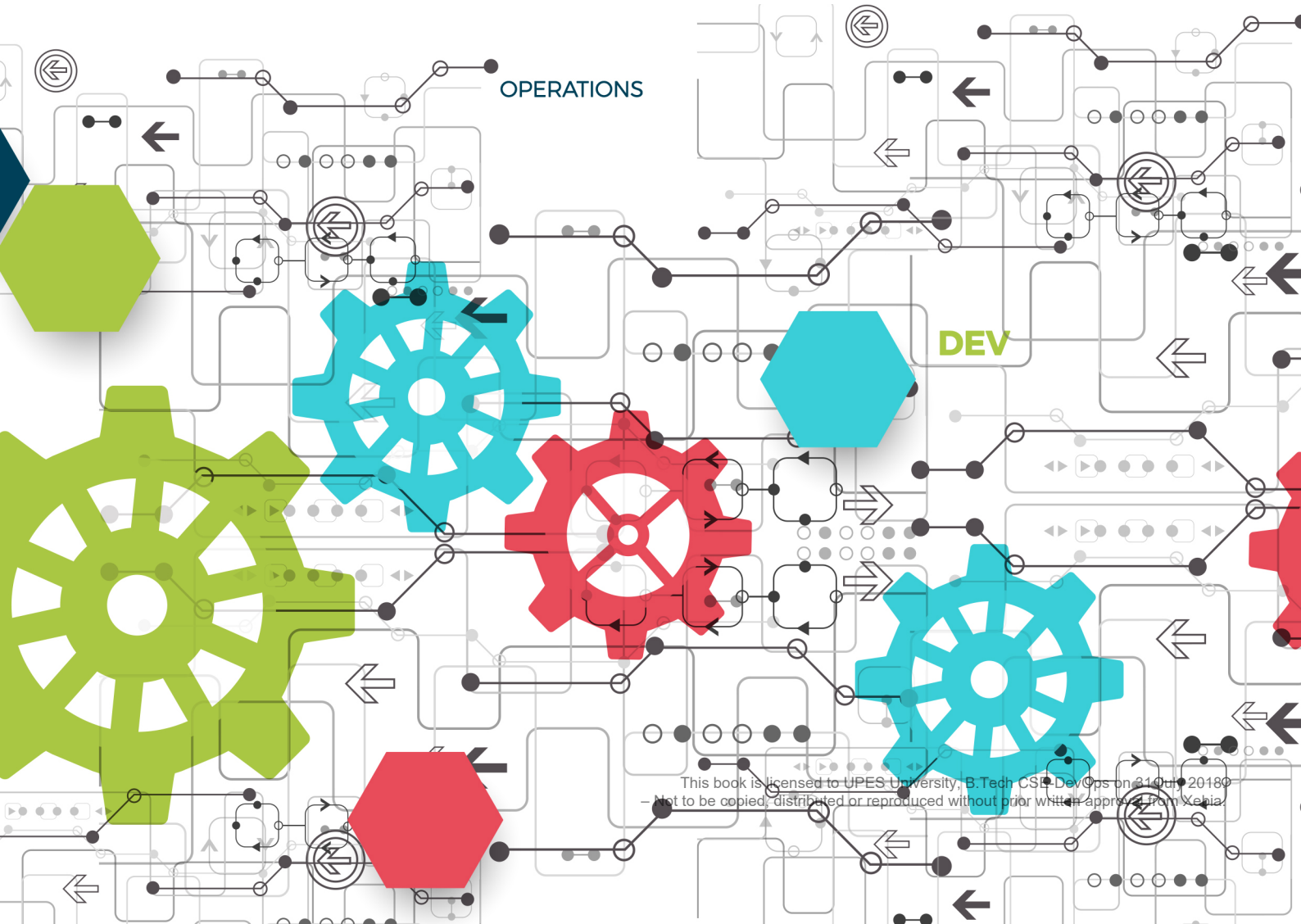
Xebia Group consists of seven specialized, interlinked companies: Xebia, Xebia Academy, XebiaLabs, StackState, GoDataDriven, Xpirit and Binx.io. With offices in Amsterdam and Hilversum (Netherlands), Paris, Delhi, Bangalore and Boston, we employ over 700 people worldwide. Our solutions address digital strategy; agile transformations; DevOps and continuous delivery; big data and data science; cloud infrastructures; agile software development; quality and test automation; and agile software security.

ODW is dedicated to provide innovative and creative solutions that contribute in growth of emerging technologies. As a learning experience provider, ODW strengths include providing unique, up to date content by combining industry best practices with leading edge technology. ODW delivers high quality solutions and services which focus on digital learning transformation.

B.Tech Computer Science and Engineering in DevOps

DEVOPS OVERVIEW

MODULE 1
Traditional Software Development



Contents

MODULE 1

Traditional Software Development

Module Learning Objectives

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

- Software and its types
- The history of software engineering and how it has evolved over time.
- Different methods of software development
- Waterfall method of software development, various stages involved in it, and the advantages and disadvantages of waterfall method
- How conflicts between the development and operations teams impact product development

1. Software

- A software is an organized information in the form of operating systems, programs, utilities and applications that enable a computer to work.
- Any set of machine-readable instructions that directs a computer's processor to perform specific operations.
- 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.

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?

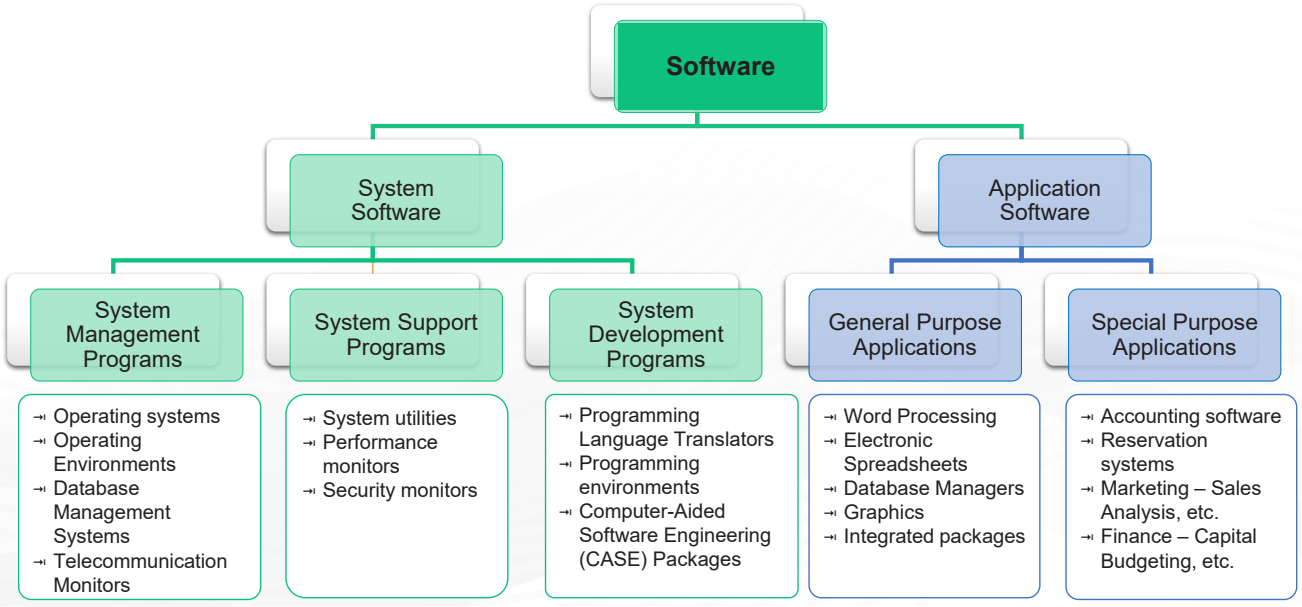
A software can be defined as an organized information in the form of operating systems, programs, utilities and applications that enable a computer to work. A software consists of carefully-organized instructions and code written by programmers in any of the different programming languages. A software 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). 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, 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).

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 about 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 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

- 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 are 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 for special purpose software include:

- Accounting software
- Reservation systems
- Marketing – Sales Analysis, etc.
- Finance – Capital Budgeting, etc.

Software Engineering: Definition

"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

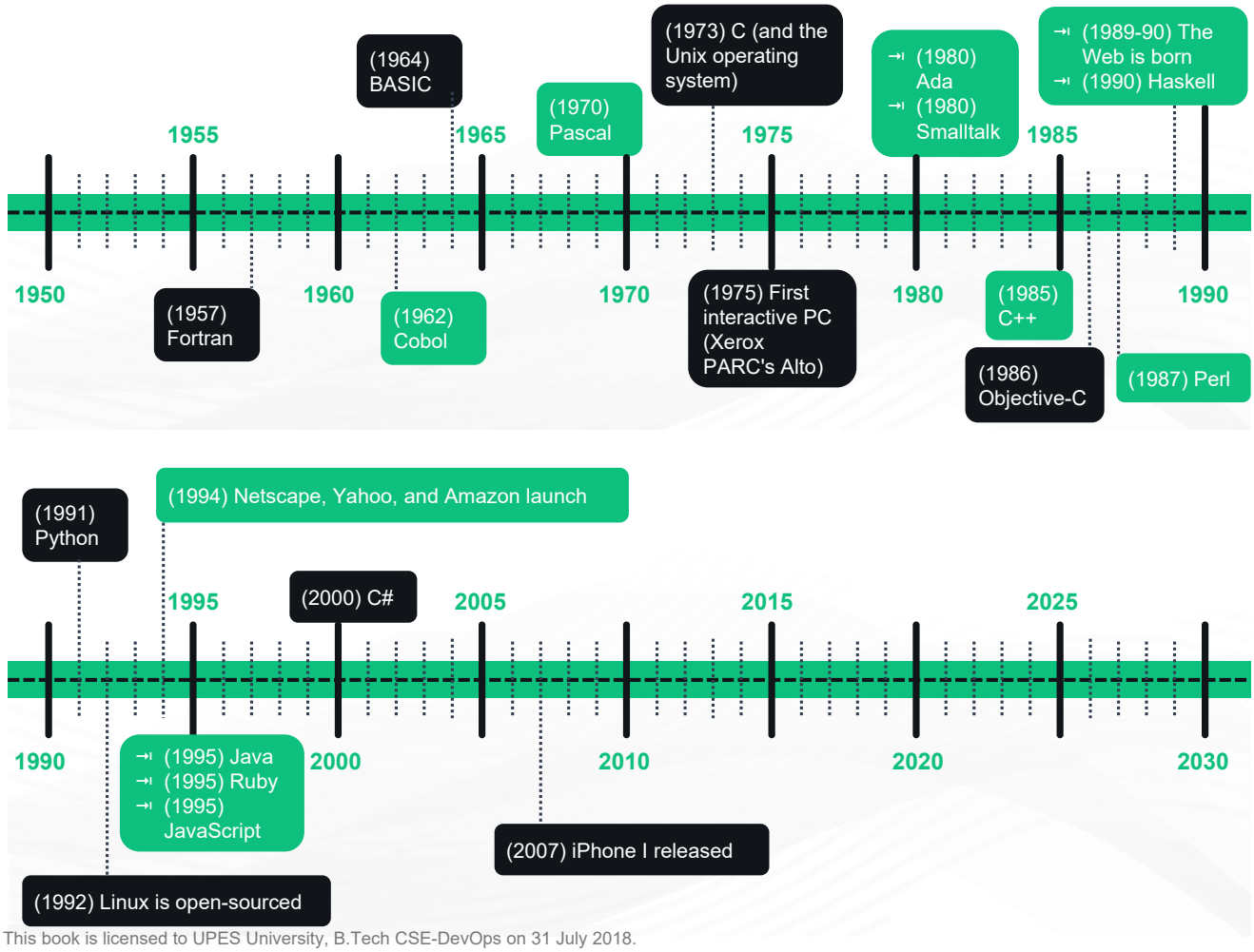
Software engineering is a detailed study of engineering to the design, development and maintenance of software. Software is important because it affects nearly every aspect of our lives and has become pervasive in our commerce, culture and everyday activities. Software engineering is important because 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.

1.1 History of Software Engineering



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 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, 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 in 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 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 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.

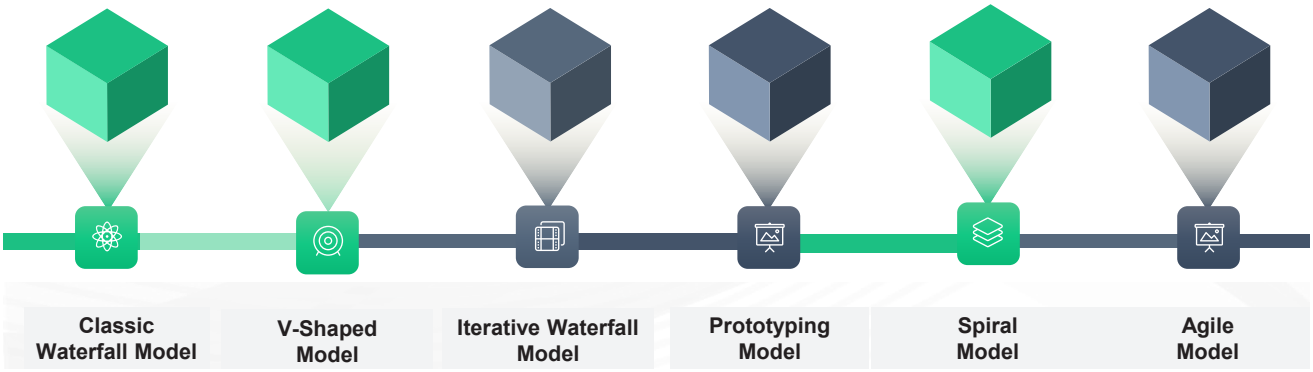
Mobile Computing

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.

1.3 Different Software Life Cycle Models

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



Many software life cycle models have been proposed so far. Each model comes with its own set of advantages or disadvantages. A few important and commonly used life cycle models are as follows:

1. 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:

- Requirements
- Analysis
- Design
- Coding
- Testing
- Operations/Maintenance

2. 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 if time. This model suits well for small projects, where requirements are clearly defined and easily understood. The steps involved in this model are:

- Requirements
- System design
- Architecture design
- Module design
- Implementation/Coding
- Unit testing
- Integration testing
- System testing
- Acceptance testing

3. Iterative Waterfall Model

Unlike the more traditional/classical waterfall model, which focuses on a stringent step-by-step process of development stages, the iterative model is best thought of as a cyclical process. After an initial planning phase, a small handful of stages are repeated over and over, with each completion of the

cycle incrementally improving and iterating on the software. Enhancements can quickly be recognized and implemented throughout each iteration, allowing the next iteration to be at least marginally better than the last. This model can consist of mini waterfalls. The steps involved in this model are:

- Planning and requirements
- Analysis and design
- Implementation
- Testing
- Evaluation

4. 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. 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.

5. 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

6. Agile Model

Unlike other SDLC models, Agile focuses less on specific requirements or guidelines, and far more on abstraction of these best practices to allow for greater flexibility, or agility, during the development process. Agile model emphasizes the need for every project to be handled differently, based on the individual needs of the project, the schedule, and the team behind it. You will learn about agile development in detail in the upcoming modules.

What did you Grasp?

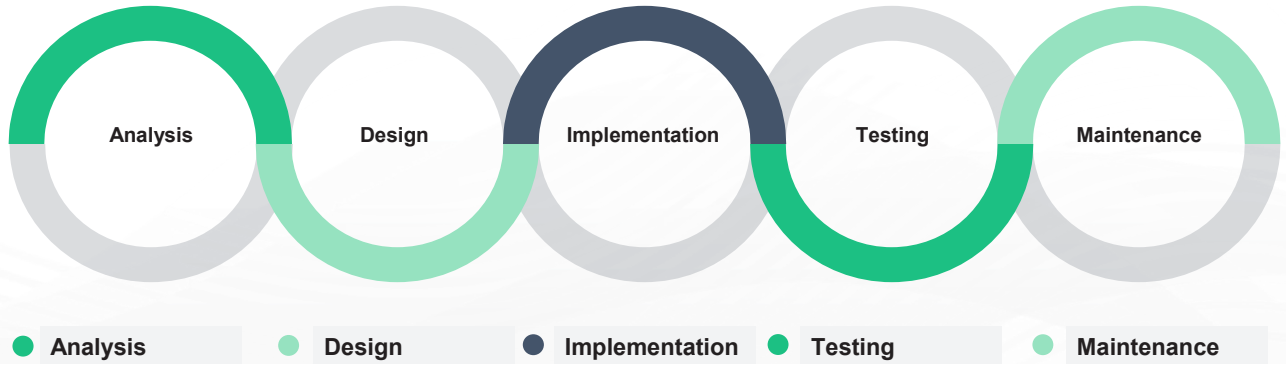
1. The process of developing a software product using software engineering principles and methods is referred to as, _____

   A) Software myths
   B) Scientific Product
   C) Software Evolution
   D) None of the above

2. What is the main aim of Software engineering?

   A) Reliable software
   B) Cost effective software
   C) Reliable and cost effective software
   D) None of the above

2. Waterfall Model



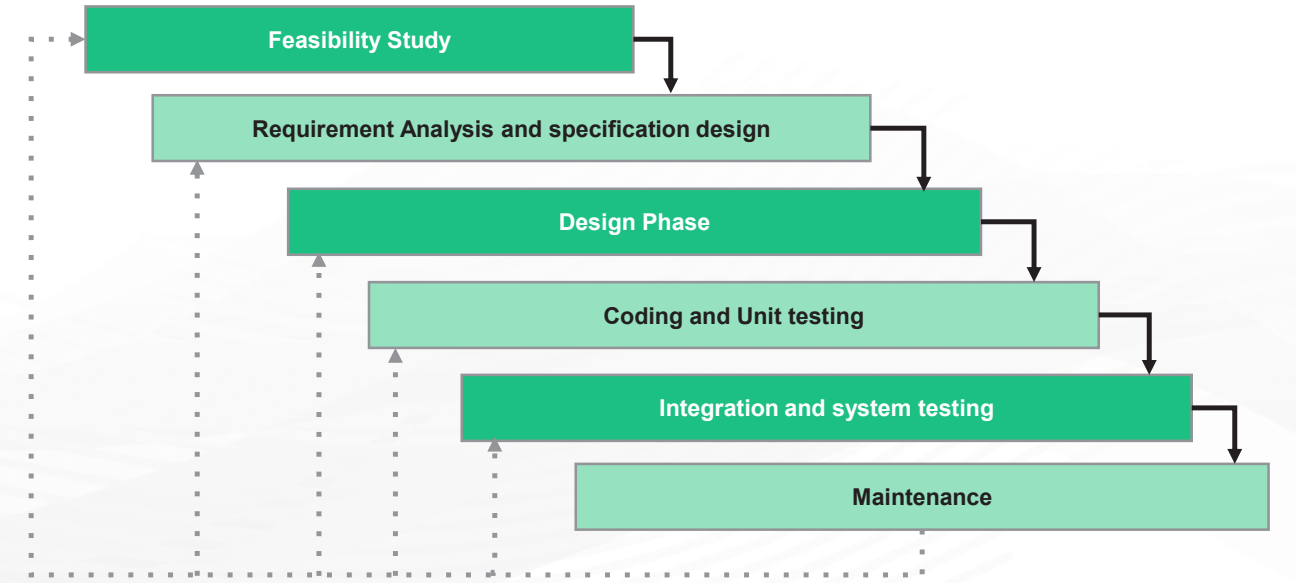
The most popular way of developing a software is the 'THE CLASSICAL WATERFALL' model . It is the first SDLC model ,introduced to describe the software development ,in late 1950 and became popular in 1970's by the first formal description of the waterfall model cited in an article by Winston W. Royce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

One of the main requirements of Software development process is the non-iterative sequential WATERFALL MODEL. The entire software development process undergoes different phases, namely requirement details/analysis, design, coding & testing, system testing & integration and maintenance respectively.

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

2.1 Explanation of Classical Model of Waterfall

In “The Waterfall” approach, the outcome of one phase acts as the input for the next phase sequentially



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.

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

2.1.1 First Stage – Feasibility Study

The following activities are covered during the feasibility study phase.

1. Financial Feasibility
2. Technical Feasibility
3. Client Visit
4. Study Of Input-output Data
5. Case Study

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. {????????}

2.1.2 Second Stage – Requirements Analysis & Specification Design

The following activities are carried out during the requirements analysis and specification design phase.

1. Requirements gathering and analysis
2. Requirements specification

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 e.g., 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. Therefore to resolve them through further discussions with the customer.

Once all ambiguities, inconsistencies, and incompleteness have been resolved and all the requirements/ expectations have been properly understood, the **requirements specification** activity can start.

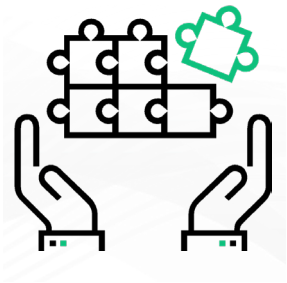
This activity includes the steps to systematically capture and organize the user requirements into a ''Software/hardware Requirements Specification''(SRS) document.

The customer requirements identified during the requirements gathering and analysis activity are organized into a SRS document. The important components of this document are f**unctional requirements, the nonfunctional requirements, and the goals of implementation**.

2.1.3 Third Stage – Design Phase

The following activities are covered during the design phase.

1. Creation of system design as per the requirements gathered during the previous phase.
2. Definition of overall system architecture
3. Documentation of design



The two Design Phases

- High level design
- Low level design

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:

1. High level design

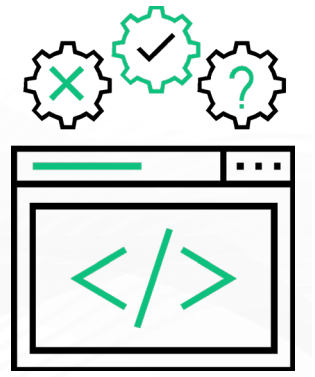
High-Level Design (HLD) involves decomposing a system into modules, and representing the interfaces and invocating relationships among modules. An HLD is referred to as software architecture. A HLD document will usually include a high-level architecture diagram depicting the components, interfaces, and networks that need to be further specified or developed. The document may also depict or otherwise refer to work flows and/or data flows between component systems.

2. 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. 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.

2.1.4 Fourth Stage – Coding and Unit Testing



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

1. Implementation Phase
2. Developing Source Code
3. Program Modules
4. Testing Of Modules
5. Debug The Errors

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 is 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.

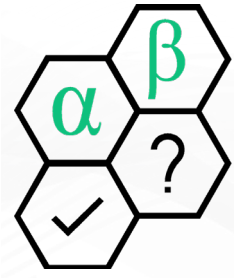
2.1.5 Fifth Stage – Integration and System Testing

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

1. Integration in Planned Manner
2. Steps of Integration
3. Addition of Modules
4. System Testing

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** are 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



1. Alpha (α) - Testing (Development team)
2. Beta (β) - Testing (set of customers)
3. Acceptance Testing (customers)

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, ALM. Finally the progress report of testing activities is prepared.

2.1.6 Sixth Stage – System Deployment and Maintenance

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

1. Product Deployment
2. Development and Maintenance

Maintenance

- Corrective Maintenance
- Perfective Maintenance
- Adaptive Maintenance

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 efforts 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.

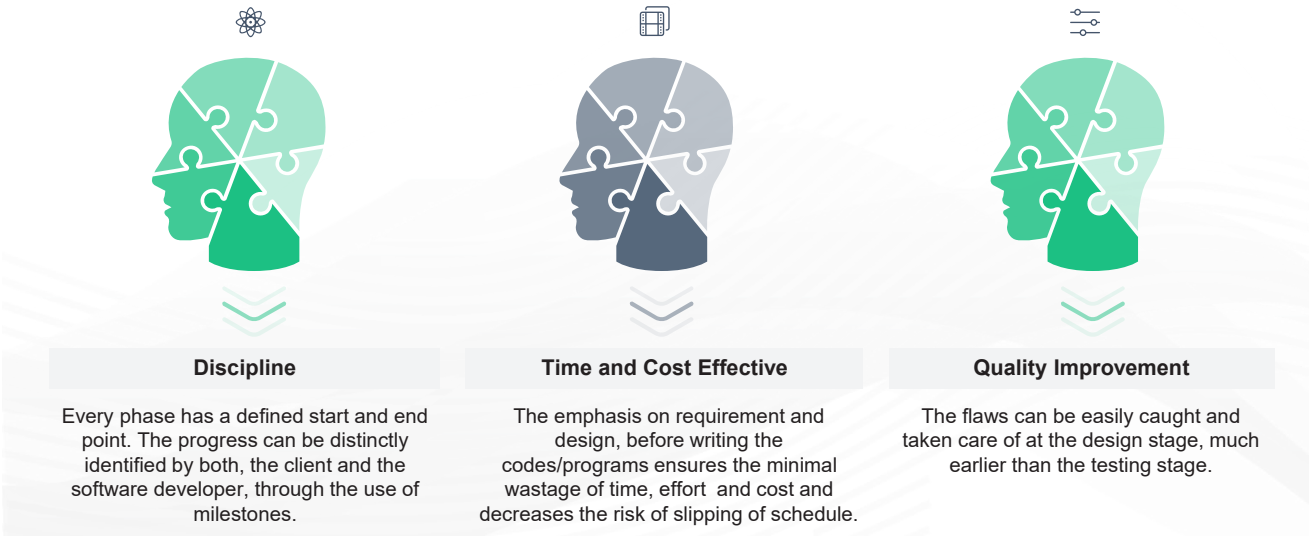
Maintenance involves performing any one or more of the following three kinds of activities:

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.

2.2 Advantages Of Waterfall Model



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 requirement 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.

2.3 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**

Shortcomings of waterfall model

It is assumed that no developmental error is ever committed by the engineers during any of the life cycle phases in the classical 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: due to oversight ,wrong assumptions, use of inappropriate technology, communication gap among the project engineers, etc.

- 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 classical 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 product can be lead to roadblock at a very later stage.
- Lack of adaptability across all stages of development is the most daunting disadvantage of the waterfall model. When a test in the fifth stage reveals 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 into 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 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?

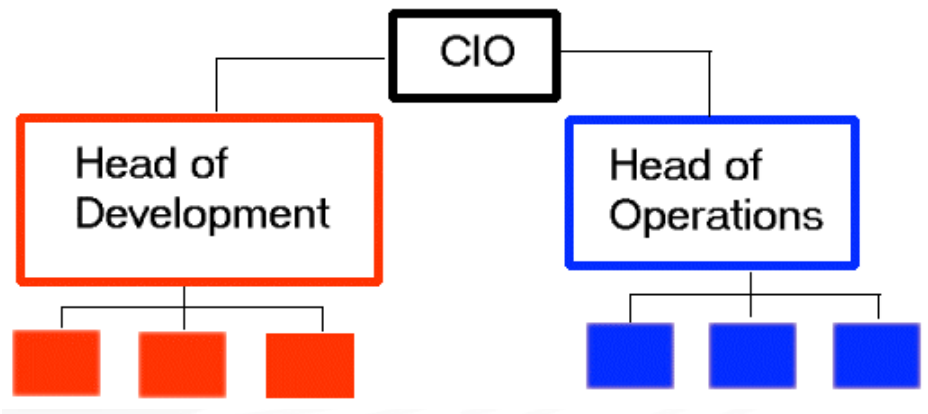
1. Which model is also called as the classic life cycle or the Waterfall model?

   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

Traditional IT Organizations

- Cultural hindrance between Development and Operations teams in traditional SDLC.
- Development and Operations - two sides of an equation - holding their own roles and responsibilities.
- Development team works independently on code.
- Code sent to testing team for validation.
- Operations team comes in toward the end of the process, handover of the release.
- No collaboration between teams.



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 be driven by 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. Operation team comes in toward the end of the process, where handover of release is given.

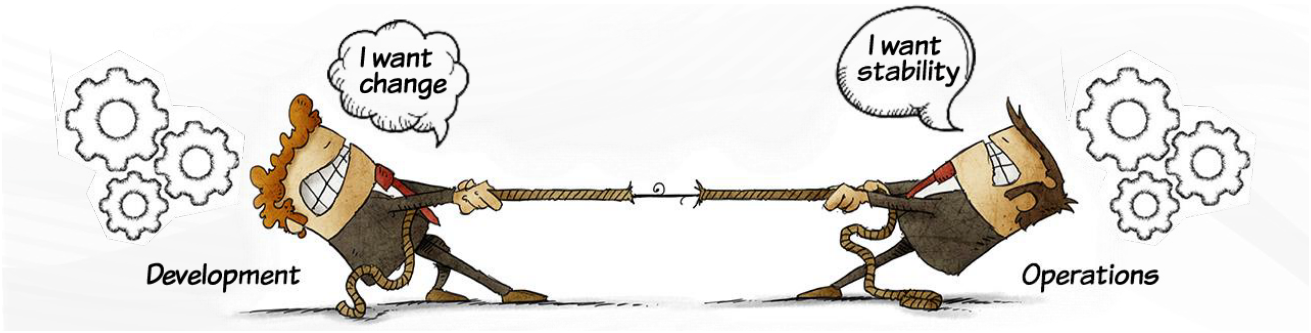
Most of the organizations, who have adopted traditional SDLC, face a situation like this on a daily basis:

- 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.

3. Developers vs IT Operations Conflict

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



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 hand over 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.

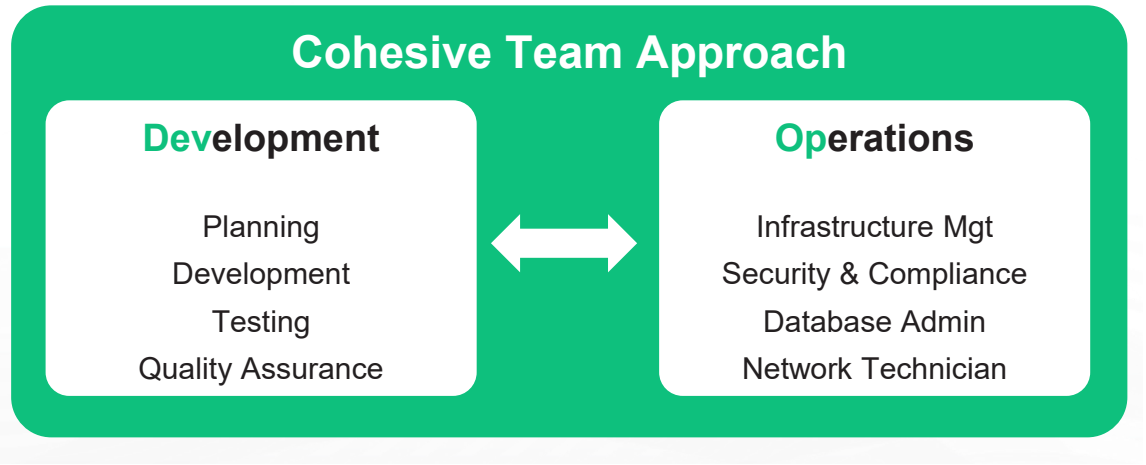
In the absence of required discussions between the Development and the Operations teams during earlier phases of development, a lot of useful information is not shared between the two teams. Such information is crucial for the Operations team to get ready for the upcoming changes to the applications 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 to solve known problems. In the presence of the wall of confusion, these elaborate knowledge articles are missed. 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 into conflict and inefficiency which is generally a mindset called wall of confusion.

3.1 Problems with the Traditional Development and the Operations

1. Organizational Silos
2. Different Mindsets
3. Different Implementations
4. Different Tools
5. Lack of Interest in Learning Other Tools
6. Different Environments
7. Loss of Work
8. Blame Game
9. Build Rollback
10. Disintegrated Process
11. No Feedback Loop



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 changes 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 teams 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, Development team might deploy to a Test environment using 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 artifacts, called build artifacts that can be run on computer systems. A build doesn't always go right. Many times teams will be forced to rollback 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 instill those same values and practices into Operations. You will learn in detail about DevOps and Agile 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

In a nutshell, we learnt:

1. The definitions available for Software and types of software.
2. The history of software engineering and how it has evolved over time.
3. Different methods of software development
4. Waterfall method of software development, various stages involved in it, and the advantages and disadvantages of waterfall method
5. How conflicts between the development and operations teams impact product development
6. A brief of DevOps and agile as a solution for resolving the issues between development and operations teams.

Release Notes

B. TECH CSE with Specialization in DevOps

Semester One -Year 01

Release Components.
Facilitator Guide, Facilitator Course Presentations, Student Guide and Mock exams.

Current Release Version.
1.0.0

Current Release Date.
2 July 2018

Course Description.
Xebia, has been recognized as a leader in DevOps by Gartner and Forrester and this course is created by Xebia to equip students with set of practices, methodologies and tools that emphasizes the collaboration and communication of both software developers and other information-technology (IT) professionals while automating the process of software delivery and infrastructure changes.

The language used in this course is US English. Our sources of reference for grammar, syntax, and mechanics are from The Chicago Manual of Style, The American Heritage Dictionary, and the Microsoft Manual of Style for Technical Publications.

Bugs reported	Not applicable for version 1.0.0
Next planned release	Version 2.0.0 Feb 2019