SOFTWARE ENGINEERING

UNIT – 1

CHAPTER – 1

INTRODUCTION TO SOFTWARE ENGINEERING

I. Definition

Software is program/ set of programs containing instructions which provide desired functionality. Also comprises of data structures that enable the program to manipulate information

Engineering is the process of designing and building something that serves a particular purpose

Software Engineering is a systematic approach to the development, operation and maintenance of desired software

II. Evolving Role of Software

Software serves as a:

- Product: It delivers the computing potential of a H/W i.e., enables the h/w to deliver the expected functionality. Acts as information transformer
- Vehicle to deliver the product: Helps in creation and control of other programs i.e., it helps other software to do functions and helps as platform. E.g., Operating System

Today, software takes on a dual role. It is a product and, at the same time, the vehicle for delivering a product. As a product, it delivers the computing potential embodied by computer hardware or, more broadly, a network of computers that are accessible by local hardware. Whether it resides within a cellular 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. 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). Software delivers the most important product of our time—information. Software transforms personal data (e.g., an individual's financial transactions) so that the data can be more useful in a local context; it manages business information to enhance competitiveness; it provides a gateway to worldwide information networks (e.g., Internet) and provides the means for acquiring information in all of its forms. The role of computer software has undergone significant change over a time span of little more than 50 years. Dramatic improvements in hardware performance, profound changes in computing architectures, vast increases in memory and storage capacity, and a wide variety of exotic input and output options have all precipitated more sophisticated and complex computer-based systems. The lone programmer of an earlier era has been replaced by a team of software specialists, each focusing on one part of the technology required to deliver a complex application.

III. Why Software Engineering is important?

i. Imposes discipline to work that can become quite chaotic - Lot of steps are involved in the development of a s/w, so if a systematic approach is not taken, it becomes difficult/clumsy

ii. Ensures high quality of software - If a s/w could deliver the features and functionalities required then it is a high-quality software.

iii. Enables us to build complex systems in a timely manner - Whenever we have a huge/complex project, then we need to set proper deadlines and milestones of the time taken in each step, so that in time we can deliver the s/w to the customer.

IV. Difference between Software and Hardware

Software:

- S/W is logical unit
- No spare parts for s/w
- Problem statement may not be complete and clear initially
- Requirements may change with time
- Multiple copies (less cost)
- Idealized and actual graph

Hardware:

- H/W is physical unit
- Spare parts for h/w exist

- Problem statement is clearly mentioned at the beginning of development
- Requirements are fixed before manufacturing
- Multiple copies (more cost)
- Bath tub graph

Bath Tub Curve:

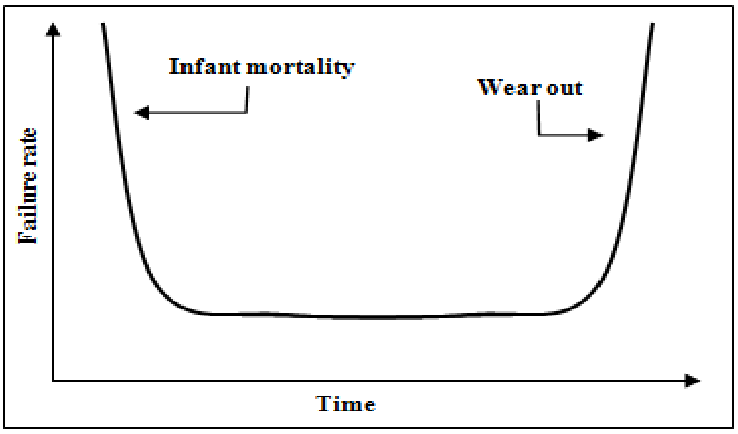


Figure 1: Bath tub Curve- Failure curve for Hardware

- Given the name bath tub because its shape is same as bath tub.
- It explains the reliability of the product i.e. until how many days / time the product works
- We have 3 stages in bath tub curve:
  - Decreasing failure rate or Infant mortality
  - Constant failure rate
  - Increasing failure rate or wear out (H/W effected by the environment factors like dust, temperature, pollution, etc). S/W does not have wear out.
- Decreasing failure rate:
  - As the product in this stage is new, there are very less chances of it to fail. The product still fails because of
    - Manufacturing defect
    - We haven't assembled it properly

  - It is a weak part or product.

- Constant failure rate:
  - It is named as constant failure rate because the graph remains in straight line according to the time.
  - Most of the products fail in this stage
  - It is the service life of the product.
- Increasing failure rate:
  - This is the stage where we use the product more than its service period. The suddenly, the product may fail.
  - It is named increasing failure rate as the curve moves upwards as shown according to the time.

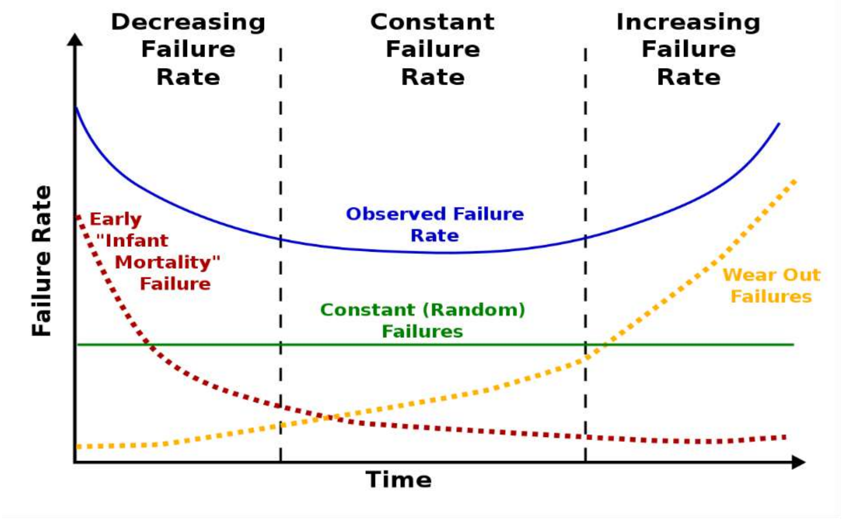


Figure 2: Failure rates in Bath tub Curve

Idealized Curve:

- Since there is no wear out in s/w the curve must undergo on 2 phases i.e., decreasing failure rate and constant failure rate which is called as Idealized curve.
- But in reality, idealized curve is not possible.
- Initial failure happens just like h/w due to undetected defects.

- After correcting the defects, the curve reaches a steady state, but if any change occurs in the s/w then there is spike in the graph.
- Most of the times the failure rate increases when a change effect is requested. The actual curve is higher than the idealized curve.

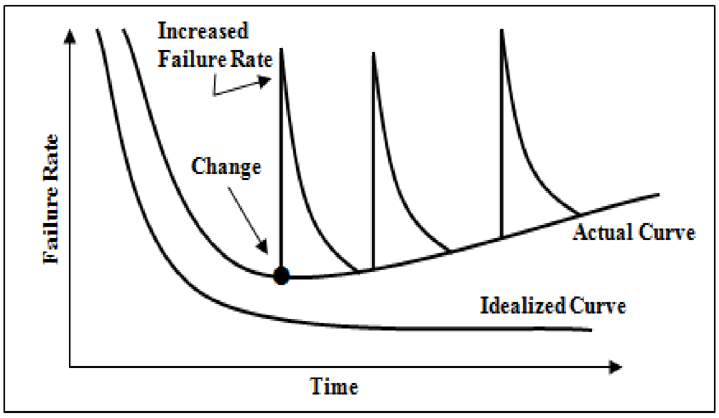


Figure 3: Idealized Curve- Failure curve for Software

V. Work Product:

Work product is the result or outcome of Software Engineering

The outcome is in 2 perspectives:

1. Software Engineer: The set of programs, the content (data) along with documentation that is a part of s/w.

2. User/Customer: The functionality delivered by the s/w that improves user experience

VI. Software Engineering focuses on:

- Quality (While Software development)

– Functional: To what extent that we have delivered the correct s/w and is it reaching the expectations. Degree to which correct software is produced.

– Non functional: Also called structural attributes. Features other than functions of s/w like robustness, security, etc.

- Maintainability (After the software has been developed and delivered)

- Should be easily enhanced and adapted to changing requirements whenever required.

VII. Changing Nature of Software:

The nature of software has changed a lot over the years.

1. **System software:** Infrastructure software come under this category like compilers, operating systems, editors, drivers, etc. Basically, system software is a collection of programs to provide service to other programs.
2. **Real time software:** This software is used to monitor, control and analyse real world events as they occur. An example may be software required for weather forecasting. Such software will gather and process the status of temperature, humidity and other environmental parameters to forecast the weather.
3. **Embedded software:** This type of software is placed in “Read-Only- Memory (ROM)” of the product and control the various functions of the product. The product could be an aircraft, automobile, security system, signalling system, control unit of power plants, etc. The embedded software handles hardware components and is also termed as intelligent software.
4. **Business software:** This is the largest application area. The software designed to process business applications is called business software. Business software could be payroll, file monitoring system, employee management, account management. It may also be a data warehousing tool which helps us to take decisions based on available data. Management information system, enterprise resource planning (ERP) and such other software are popular examples of business software.
5. **Personal computer software:** The software used in personal computers are covered in this category. Examples are word processors, computer graphics, multimedia and animating tools, database management, computer games etc. This is a very upcoming area and many big organisations are concentrating their effort here due to large customer base.
6. **Artificial intelligence software:** Artificial Intelligence software makes use of non-numerical algorithms to solve complex problems that are not amenable to computation or straight forward analysis. Examples are expert systems, artificial neural network, signal processing software etc.

7. **Web based software:** The software related to web applications come under this category. Examples are CGI, HTML, Java, Perl, DHTML etc.

VIII. Legacy Software:

Legacy:

- Generally, something which u get in inheritance
- Example: Money / Status
- In terms of s/w, something which is old but still in use and difficult to replace.
- Example: Legacy car

Legacy System:

- The combination of legacy software and hardware overall is called legacy system.
- Ex: a s/w working in win XP is not compatible with windows 11.

Legacy Software:

- Outdated s/w still in use as it is fulfilling the needs of the organization
- Ex: Bank s/w
- Customize s/w: s/w once developed will not have any changes or new requirements then we use it for decade
- Ex: Games

Need to be replaced for the following reasons:

Due to change in business model

Due to change in architecture

Inadequate security, performance, flexibility, etc.

IX. Software Myths:

The development of software requires dedication and understanding on the developers' part. Many software problems arise due to myths that are formed during the initial stages of software development. Software myths propagate false beliefs and confusion in the minds of management, users and developers. Blind belief that management, customers and practitioners have on s/w and the process to use it. Managers, who own software development responsibility, are often under strain and pressure to maintain a software budget, time constraints, improved quality, and many other considerations. Common management myths are:

- Management myths
- Customer myths
- Practitioner's myths

i. **Management Myths:**

Myth 1:

We have all the standards and procedures available for software development.

<u>Fact</u>:

- Software experts do not know all the requirements for the software development.
- And all existing processes are incomplete as new software development is based on new and different problem.

Myth 2:

The addition of the latest hardware programs will improve the software development.

<u>Fact</u>:

- The role of the latest hardware is not very high on standard software development; instead (CASE) Engineering tools help the computer, they are more important than hardware to produce quality and productivity.
- Hence, the hardware resources are misused.

Myth 3:

- With the addition of more people and program planners to Software development can help meet project deadlines (If lagging behind).

<u>Fact</u>:

- If software is late, adding more people will merely make the problem worse. This is because the people already working on the project now need to spend time educating the newcomers, and are thus taken away from their work. The newcomers are also far less productive than the existing software engineers, and so the work put into training them to work on the software does not immediately meet with an appropriate reduction in work.

ii. Customer Myths:

The customer can be the direct users of the software, the technical team, marketing / sales department, or other company. Customer has myths leading to false expectations (customer) & that's why you create dissatisfaction with the developer.

Myth 1:

A general statement of intent is enough to start writing plans (software development) and details of objectives can be done over time.

Fact:

- Official and detailed description of the database function, ethical performance, communication, structural issues and the verification process are important.
- Unambiguous requirements (usually derived iteratively) are developed only through effective and continuous communication between customer and developer.

Myth 2:

Software requirements continually change, but change can be easily accommodated because software is flexible

Fact:

- It is true that software requirements change, but the impact of change varies with the time at which it is introduced. When requirements changes are requested early (before design or code has been started), the cost impact is relatively small **(Refer Figure 4)**. However, as time passes, the cost impact grows rapidly—resources have been committed, a design framework has been established, and change can cause upheaval that requires additional resources and major design modification.

iii. Practitioner's Myths:

Myths 1:

They believe that their work has been completed with the writing of the plan.

Fact:

- It is true that every 60-80% effort goes into the maintenance phase (as of the latter software release). Efforts are required, where the product is available first delivered to customers.

Myths 2:

There is no other way to achieve system quality, until it is "running".

<u>Fact</u>:

- Systematic review of project technology is the quality of effective software verification method. These updates are quality filters and more accessible than test.

  Myth 3:

  An operating system is the only product that can be successfully exported project.

<u>Fact</u>:

- A working system is not enough, the right document brochures and booklets are also required to provide guidance & software support.

  Myth 4:

  Engineering software will enable us to build powerful and unnecessary document & always delay us.

<u>Fact</u>:

- Software engineering is not about creating documents. It is about creating a quality product. Better quality leads to reduced rework. And reduced rework results in faster delivery times.

X. SOFTWARE ENGINEERING - A LAYERED TECHNOLOGY

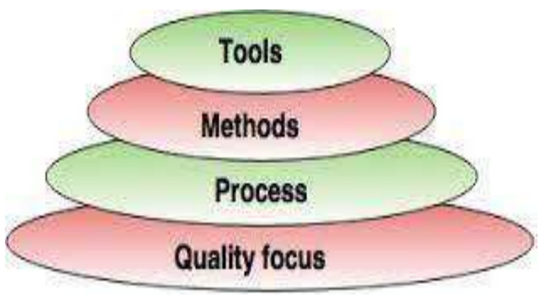
IEEE has developed a more comprehensive definition of software engineering:

"Software engineering is the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software."

- **Software Engineering is a layered technology.**

Software Engineering encompasses a **Process, Methods** for managing and engineering software and **tools**.

The following Figure represents **Software engineering Layers**



Software Engineering – A Layered Technology

Referring to the above Figure, any engineering approach must rest on an organizational commitment to **quality**. The bedrock that supports software engineering is a **quality focus**. The foundation for software engineering is the ***process*** **layer**. The software engineering process is the glue that holds the technology layers together and enables the rational and timely development of computer software. **Process** defines a **framework** that must be established for the effective delivery of software engineering technology.

Software engineering ***methods*** provide the technical **how-to** for building software. **Methods** encompass a broad array of tasks that include communication, requirements analysis, design modelling, program construction, testing, and support. Software engineering ***tools*** provide **automated or semi-automated** support for the process and the methods. When tools are integrated so that information created by one tool can be used by another, a system for the support of software development, called computer-aided software engineering, is established.

XI. THE SOFTWARE PROCESS FRAMEWORK

A ***process*** is a collection of **activities, actions, and tasks** that are performed when some work product is to be created.

An ***activity*** strives to achieve a broad objective (e.g., communication with stakeholders) and is applied regardless of the application domain, size of the project, complexity of the effort, or degree of rigor with which software engineering is to be applied.

An ***action*** encompasses a set of tasks that produce a major work product (e.g., an architectural design model).

A ***task*** focuses on a small, but well-defined objective (e.g., conducting a unit test) that produces a tangible outcome.

A ***process framework*** establishes the foundation for a complete software engineering process by identifying a small number of ***framework activities*** that apply to all software projects, regardless of their size or complexity. In addition, the process framework encompasses a set of ***umbrella activities*** that are applicable across the entire software process.

A generic process framework for software engineering encompasses **five** activities:

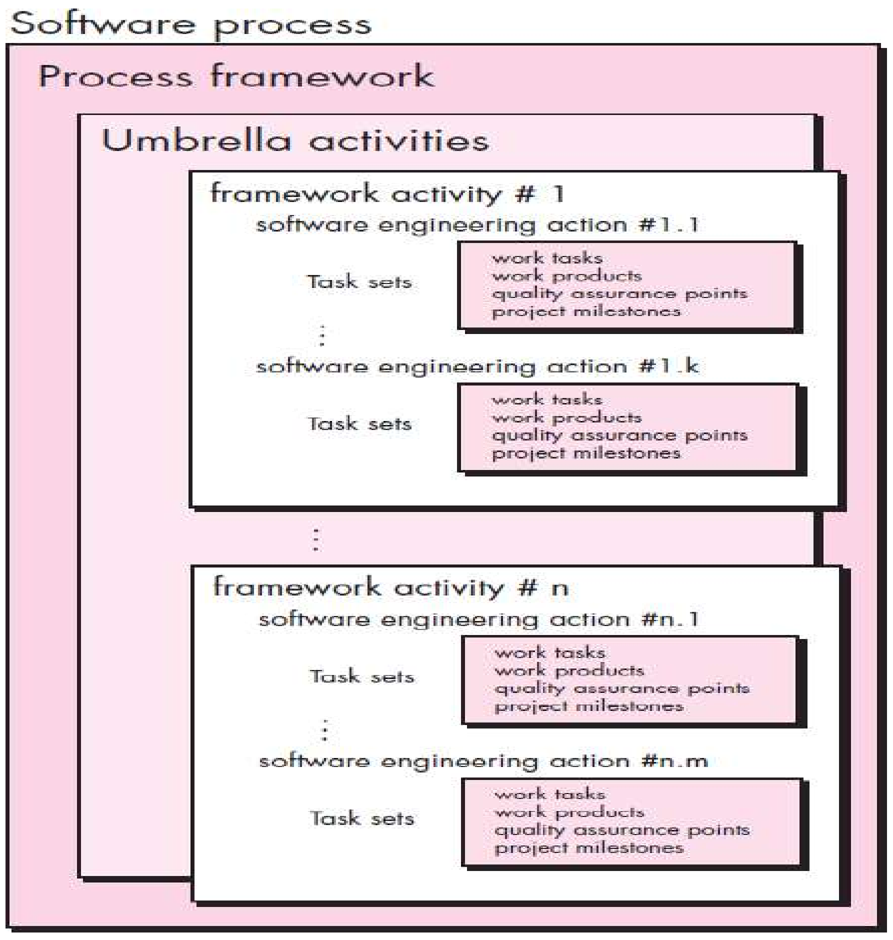
- **Communication**
- **Planning**
- **Modelling**
- **Construction**
- **Deployment**

These **five** generic framework activities can be used during the development of small, simple programs, the creation of large Web applications, and for the engineering of large, complex computer-based systems.

Software engineering process framework activities are complemented by several U**mbrella Activities**. In general, **umbrella activities** are applied throughout a software project and help a software team manage and control progress, quality, change, and risk. Typical umbrella activities include:

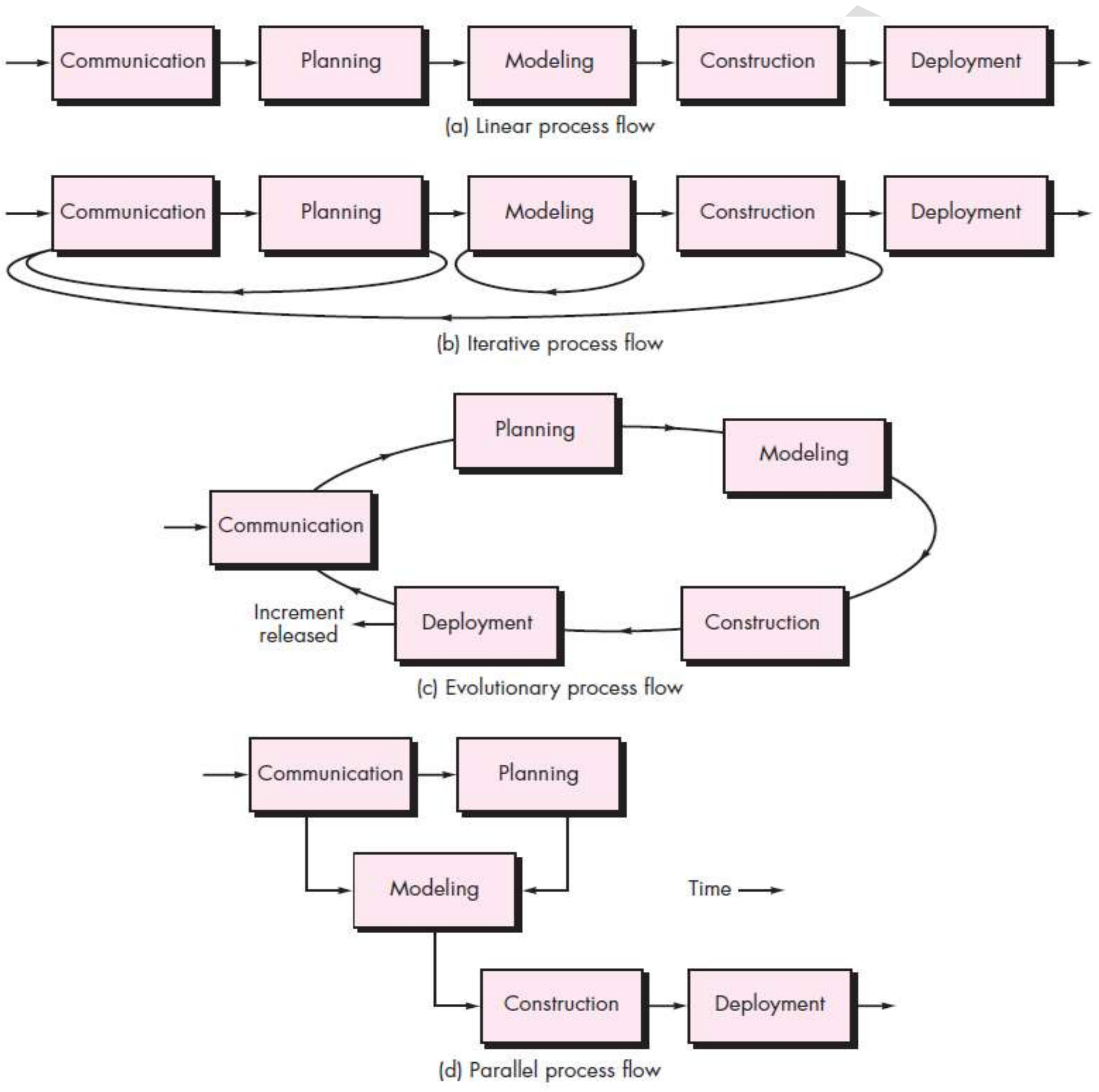
- Software project tracking and control
- Risk management
- Software quality assurance
- Technical reviews
- Measurement
- Software configuration management
- Reusability management
- Work product preparation and production

XII. A GENERIC PROCESS MODEL



The software process is represented schematically in the following figure. Each framework activity is populated by a set of software engineering actions. Each software engineering action is defined by a task set that identifies the work tasks that are to be completed, the work products that will be produced, the quality assurance points that will be required, and the milestones that will be used to indicate progress.
A generic process framework defines five framework activities—communication, planning, modeling, construction, and deployment. In addition, a set of umbrella activities like project tracking and control, risk management, quality assurance, configuration management, technical reviews, and others are applied throughout the process.

This aspect is called process flow. It describes how the framework activities and the actions and tasks that occur within each framework activity are organized concerning sequence and time and is illustrated in the following figure



A generic process framework for software engineering A linear process flow executes each of the five framework activities in sequence. An iterative process flow repeats one or more of the activities before proceeding to the next. An evolutionary process flow executes the activities in a “circular” manner. A parallel process flow executes one or more activities in parallel with other activities.

SOFTWARE ENGINEERING

UNIT – 1

CHAPTER – 2

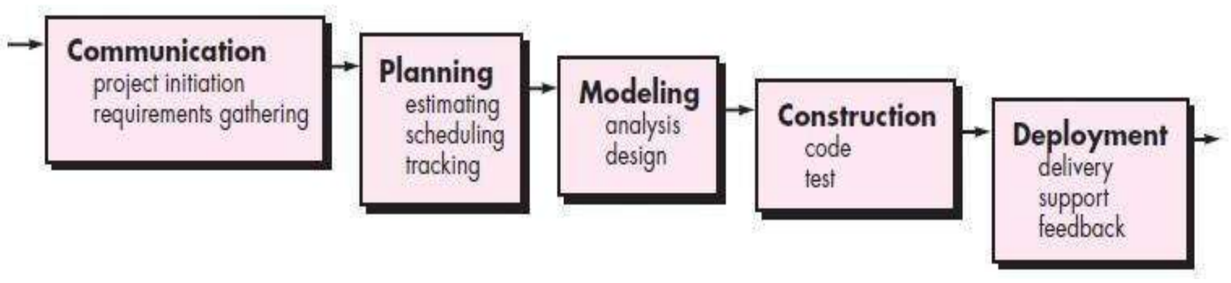
PROCESS MODELS

I. PRESCRIPTIVE PROCESS MODELS

Prescriptive process models were originally proposed to bring order to the chaos of software development. Prescriptive process models define a prescribed set of process elements and a predictable process workflow. “prescriptive” because they prescribe a set of process elements—framework activities, software engineering actions, tasks, work products, quality assurance, and change control mechanisms for each project.

II. The Waterfall Model

The **waterfall model** sometimes called the **classic life cycle**, suggests a systematic, sequential approach to software development that begins with customer specification of requirements and progresses through **planning, modeling, construction, and deployment**.



The waterfall model

The waterfall model is the oldest paradigm for software engineering. The problems that are sometimes encountered when the waterfall model is applied are:

1. Real projects rarely follow the sequential flow that the model proposes. Although the linear model can accommodate iteration, it does so indirectly. As a result, changes can confuse the project team proceeds.
2. It is often difficult for the customer to state all requirements explicitly. The waterfall model requires this and has difficulty accommodating the natural uncertainty that exists at the

beginning of many projects.

3. The customer must have patience. A working version of the program(s) will not be available until late in the project period.

This model is suitable whenever a limited number of new development efforts and when requirements are well defined and reasonably stable.

III. Incremental Process Models

The incremental model delivers a series of releases, called increments, that provide progressively more functionality for the customer as each increment is delivered.

The incremental model combines elements of linear and parallel process flows. The incremental model applies linear sequences in a staggered fashion as calendar time progresses. Each linear sequence produces deliverable "increments" of the software in a manner that is similar to the increments produced by an evolutionary process flow.

For example, word-processing software developed using the incremental paradigm might deliver basic file management, editing, and document production functions in the first increment; more sophisticated editing and document production capabilities in the second increment; spelling and grammar checking in the third increment; and advanced page layout capability in the fourth increment.

When an incremental model is used, the first increment is often a core product. That is, basic requirements are addressed but many supplementary features remain undelivered. The core product is used by the customer. As a result of use and/or evaluation, a plan is developed for the next increment. The plan addresses the modification of the core product to better meet the needs of the customer and the delivery of additional features and functionality. This process is repeated following the delivery of each increment until the complete product is produced.

Incremental development is particularly useful when **staffing (process of hiring eligible candidates) is unavailable** for a complete implementation by the business deadline that has been established for the project. Early increments can be implemented with fewer people. If the core product is well received, then additional staff (if required) can be added to implement the next increment. In addition, increments can be planned to manage technical risks.

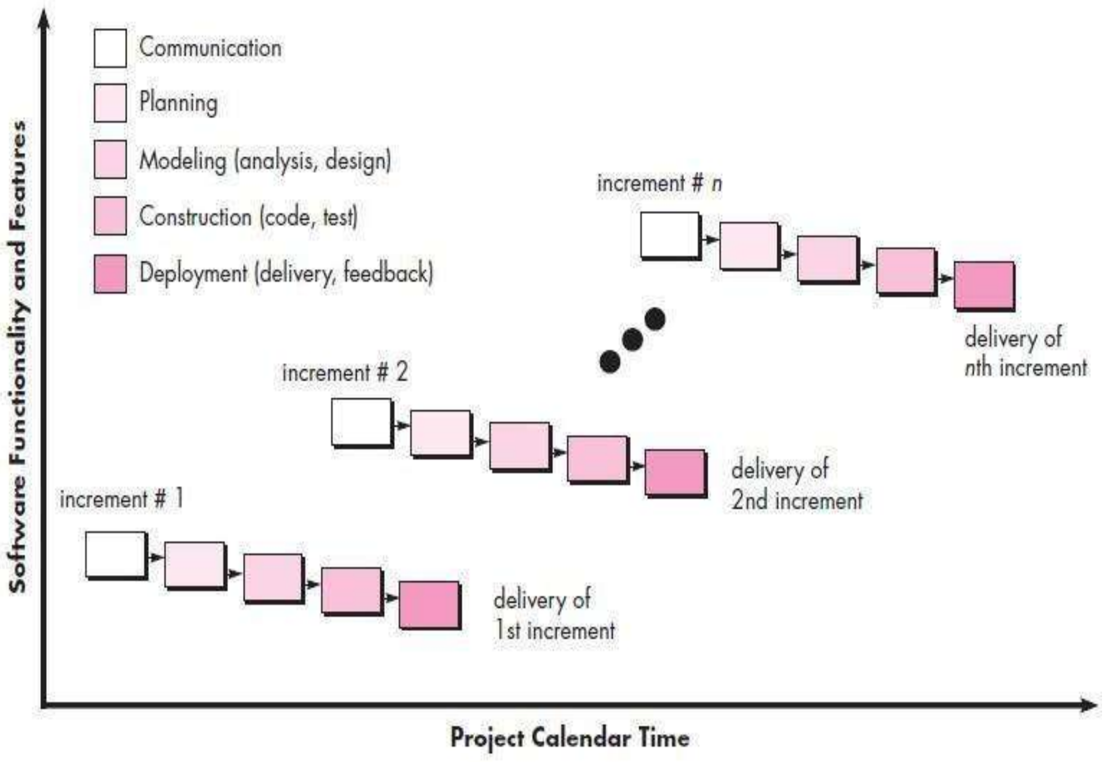

Communication
Planning
Modeling (analysis, design)
Construction (code, test)
Deployment (delivery, feedback)
increment # 1
delivery of 1st increment
increment # 2
delivery of 2nd increment
increment # n
delivery of nth increment
Software Functionality and Features
Project Calendar Time

Fig: Incremental Model

IV. Evolutionary Process Models

Evolutionary models are **iterative**. They are characterized in a manner that enables you to develop increasingly more complete versions of the software with each iteration. There are **three** common evolutionary process models.

• Prototyping Model:

Often, a customer defines a set of general objectives for software, but does not identify detailed requirements for functions and features. In other cases, the developer may be unsure of the efficiency of an algorithm, the adaptability of an operating system, or the form that human-machine interaction should take. In these, and many other situations, a **prototyping paradigm** may offer the best approach.

Although prototyping can be used as a stand-alone process model, it is more commonly used as a technique that can be implemented within the context of any one of the process models. The prototyping paradigm begins with **communication**. You meet with other stakeholders to define the overall objectives for the software, identify whatever requirements are known, and outline areas where a further definition is mandatory. A prototyping iteration is planned

quickly, and **modeling** (in the form of a "quick design") occurs. A **quick design** focuses on a representation of those aspects of the software that will be visible to end users.

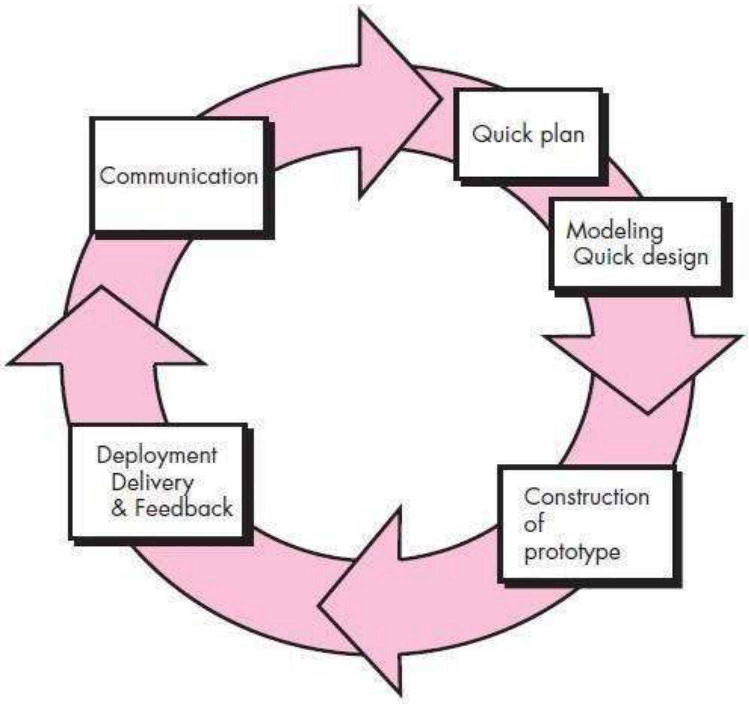


Fig: prototyping paradigm

The quick design leads to the **construction of a prototype**. The prototype is deployed and evaluated by stakeholders, who provide feedback that is used to further refine requirements.

Iteration occurs as the prototype is tuned to satisfy the needs of various stakeholders, while at the same time enabling you to better understand what needs to be done.

The prototype serves as a mechanism for identifying software requirements. If a working prototype is to be built, you can make use of existing program fragments or apply tools that enable working programs to be generated quickly. The prototype can serve as **"the first system**." Prototyping can be **problematic** for the following reasons:

1. Stakeholders see what appears to be a working version of the software, unaware that the prototype is held together haphazardly, unaware that in the rush to get it working you haven't considered overall software quality or long-term maintainability.

2. As a software engineer, you often make implementation compromises to get a prototype working quickly. An inappropriate operating system or programming language may be used simply because it is available and known; an inefficient algorithm may be implemented simply to demonstrate capability.

Although problems can occur, prototyping can be an **effective paradigm** for software engineering.

• The Spiral Model:

Originally proposed by **Barry Boehm**, the spiral model is an evolutionary software process model that couples the iterative nature of prototyping with the controlled and systematic aspects of the waterfall model. It provides the potential for rapid development of increasingly more complete versions of the software.

The spiral development model is a **risk-driven process model** generator that is used to **guide multi-stakeholder concurrent engineering** of software-intensive systems. It has **two** main distinguishing features. One is a **cyclic approach** for incrementally growing a system's degree of definition and implementation while decreasing its degree of risk. The other is a set of **anchor point milestones** for ensuring stakeholder commitment to feasible and mutually satisfactory system solutions.

Using the spiral model, the software is developed in a series of evolutionary releases. During early iterations, the release might be a model or prototype. During later iterations, increasingly more complete versions of the engineered system are produced.

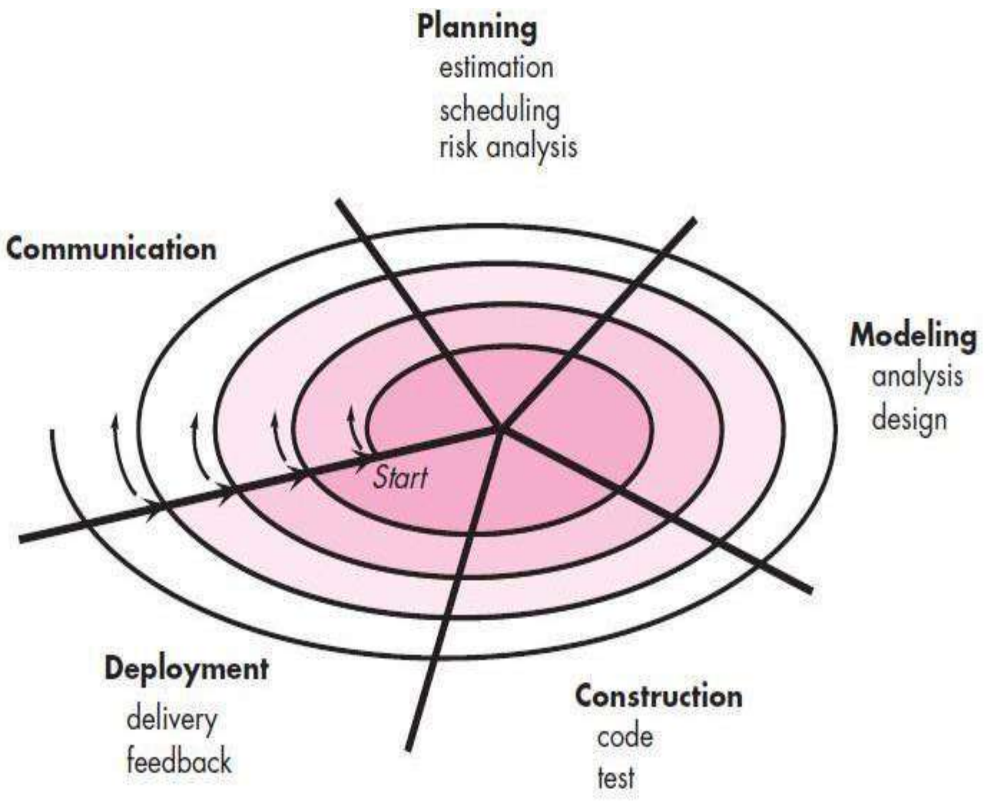


Fig: The Spiral Model

A spiral model is divided into a set of **framework activities** defined by the software engineering team. As this evolutionary process begins, the software team performs activities that are implied by a circuit around the spiral in a **clockwise** direction, beginning at the **centre**. Risk is considered as each revolution is made. **Anchor point milestones** are a combination of work products and conditions that are attained along the path of the spiral and are noted for each evolutionary pass.

The first circuit around the spiral might result in the development of a **product** specification; subsequent passes around the spiral might be used to develop a **prototype** and then progressively more sophisticated versions of the software. Each pass through the planning region results in adjustments to the project plan.

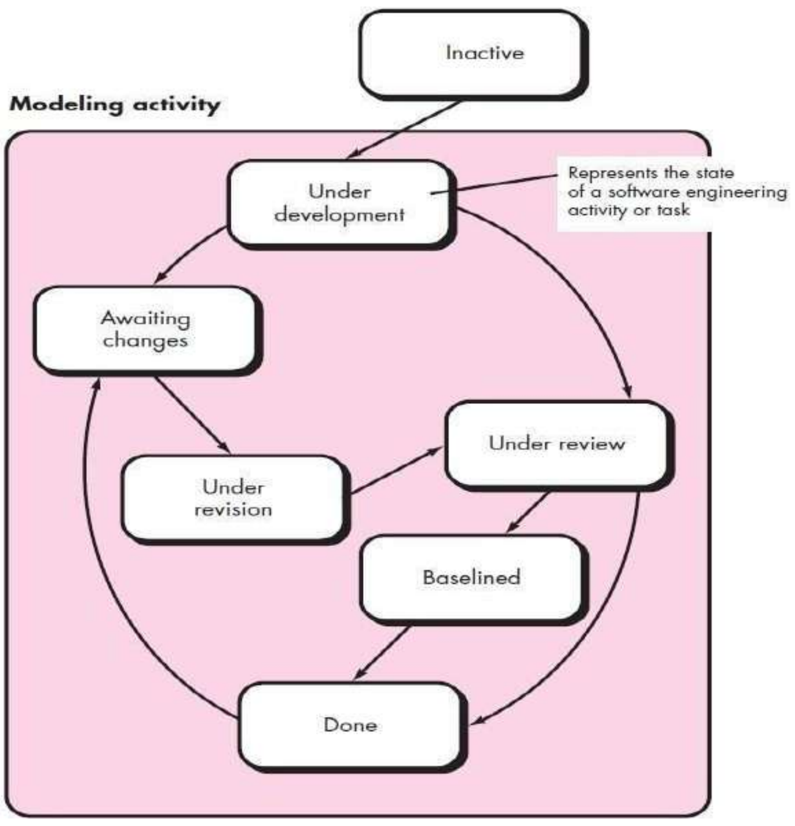
The spiral model can be adapted to apply throughout the life of the computer software. Therefore, the first circuit around the spiral might represent a "**concept development project**" that starts at the core of the spiral and continues for multiple iterations until concept development is complete. The new product will evolve through several iterations around the spiral. Later, a circuit around the spiral might be used to represent a "**product enhancement project.**"

A spiral model is a **realistic approach** to the development of **large-scale systems** and software. Because software evolves as the process progresses, the developer and customer better understand and react to risks at each evolutionary level. It maintains the systematic stepwise approach suggested by the classic life cycle but incorporates it into an iterative framework that more realistically reflects the real world.

- ## Concurrent Models:

The concurrent development model sometimes called **concurrent engineering**, allows a software team to represent iterative and concurrent elements of any of the process models. The concurrent model is often more appropriate for product engineering projects where different engineering teams are involved.

These models provide a schematic representation of one software engineering activity within the **modelling** activity using a concurrent modelling approach.



The activity **modelling** may be in any one of the states noted at any given time. Similarly, other activities, actions, or tasks (e.g., **communication** or **construction**) can be represented analogously.

All software engineering activities exist concurrently but reside in different states. Concurrent modelling defines a series of events that will trigger transitions from state to state for each of the software engineering activities, actions, or tasks. This generates the event analysis model correction, which will trigger the requirements analysis action from the **done** state into the **awaiting changes** state.

Concurrent modelling applies to all types of software development and provides an accurate picture of the current state of a project. Each activity, action, or task on the network exists simultaneously with other activities, actions, or tasks. Events generated at one point in the process network trigger transitions among the states.

SOFTWARE ENGINEERING

UNIT – 1

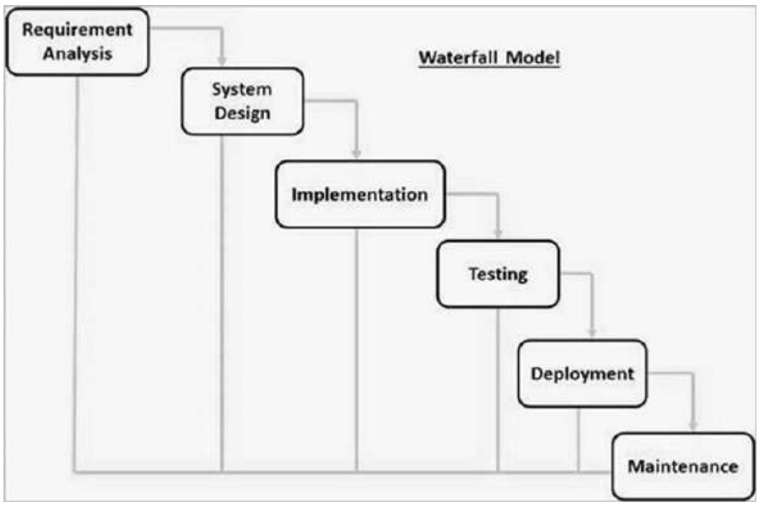
CHAPTER – 3

INTRODUCTION TO AGILE METHODOLOGY WITH SCRUM

I. AGILE VERSUS TRADITIONAL METHOD COMPARISON

- **Traditional Methodology**

The Waterfall Model was the first Process Model to be introduced. It is also referred to as a linear-sequential life cycle model. It is very simple to understand and use. In a waterfall model, each phase must be completed before the next phase can begin and there is no overlapping in the phases. The Waterfall model is the earliest SDLC approach that was used for software development.



Requirement Gathering and analysis − All possible requirements of the system to be developed are captured in this phase and documented in a requirement specification document.

System Design − the requirement specifications from first phase are studied in this phase and the system design is prepared. This system design helps in specifying hardware and system requirements and helps in defining the overall system architecture.

Implementation − with inputs from the system design, the system is first developed in small programs called units, which are integrated in the next phase. Each unit is developed and tested for its functionality, which is referred to as Unit Testing.

Integration and Testing − All the units developed in the implementation phase are integrated into a system after testing of each unit. Post integration the entire system is tested for any faults and failures.

Deployment of system − Once the functional and non-functional testing is done; the product is deployed in the customer environment or released into the market.

Maintenance − There are some issues which come up in the client environment. To fix those issues, patches are released. Also, to enhance the product some better versions are released. Maintenance is done to deliver these changes in the customer environment.

Advantages

a. It's very simple and easy to implement
b. Best suitable for small projects
c. Best suitable if requirements are fixed

Limitations

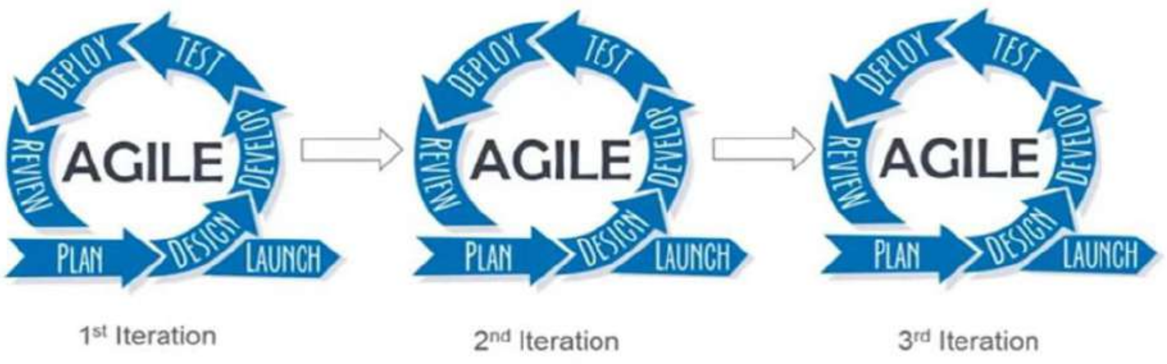
a. Development time will increase
b. Cost of development increase
c. It won't accept requirement changes in the middle
d. Client satisfaction is very low
e. Bug fixing is very costly because we can't identify bugs in early stages of life cycle
f. Not at all suitable if requirements keep on changing
g. Not suitable for large projects

II. Agile Methodology

Agile software development refers to a group of software development methodologies based on iterative development, where requirements and solutions evolve through collaboration between self-organizing cross-functional teams.

Agile methods or Agile processes generally promote a disciplined project management process that encourages frequent inspection and adaptation, a leadership philosophy that encourages teamwork, self-organization and accountability, a set of engineering best practices intended to allow for rapid delivery of high-quality software, and a business approach that aligns development with customer needs and company goals.

Agile development refers to any development process that is aligned with the concepts of the Agile Manifesto. The Manifesto was developed by a group of fourteen leading figures in the software industry, and reflects their experience of what approaches do and do not work for software development.



Advantages

a) Continuous delivery
b) Continuous feedback
c) Requirements changes in the middle
d) Client satisfaction is very high
e) Less development times
f) Less development cost

Limitations

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

g) Transfer of technology to new team members may be quite challenging due to lack of documentation.

● Principles behind the Agile Manifesto

The principles followed are:

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

SCRUM - AGILE METHODOLOGY

III. VARIOUS AGILE METHODOLOGIES

Agile model is divided into multiple sub models

- Rational unify process (RUP)
- Adaptive software development (ASD)

- Feature driven development (FDD)
- Crystal clear
- Dynamic software development method
- Extreme Programming
- SCRUM model

Among all these models scrum model is most popular and frequently used.

IV. Scrum Model

SCRUM is agile based model and is derived from rugby game, it is iterative model not a linear sequential model. Total software product will be developed increment by increment and each increment is called a sprint.

Scrum is a framework within which people can address complex adaptive problems, while productively and creatively delivering products of the highest possible value.

Scrum is a process framework that has been used to manage complex product development. Scrum is not a process or a technique for building products; rather, it is a framework. The Scrum framework consists of Scrum Teams and their associated roles, events, artifacts, and rules.

The Scrum Team consists of three roles, namely a ScrumMaster, a Product Owner, and the Team.

ScrumMaster

The ScrumMaster is the keeper of the scrum process. He/she is responsible for-

- making the process run smoothly
- removing obstacles that impact productivity
- organizing and facilitating the critical meetings

Product Owner

The Product Owner is responsible for maximizing the value of the product and the work of the Team. The Product Owner is the sole person responsible for managing the Product Backlog. Product Backlog management includes-

- Expressing Product Backlog items clearly.
- Ordering the Product Backlog items to best achieve goals and missions.
- Optimizing the value of the work the Team performs.
- Ensuring that the Product Backlog is visible, transparent, and clear to all, and shows what the Team will work on further.
- Ensuring that the Team understands items in the Product Backlog to the level needed.

The Product Owner is one person, not a committee. The Product Owner may represent the desires of a committee in the Product Backlog, but those wanting to change a Product Backlog item's priority must address the Product Owner.

The Product Owner's decisions are visible in the content and ordering of the Product Backlog. No one is allowed to tell the Team to work from a different set of requirements, and the Team is not allowed to act on what anyone else says. This is ensured by ScrumMaster.

The Team

The Team is self-organizing and cross-functional. That means the team comprises of analysts, designers, developers, testers, etc. as appropriate and as relevant to the project.

To develop a software product, we require all the roles and that is the essence of scrum – the team will function in collaboration. Cross-functional teams have all competencies needed to accomplish the work without depending on others not part of the team, and thus time and effort can be saved. The team model in Scrum is designed to optimize flexibility, creativity, and productivity.

The Team size should be kept in the range from five to nine people. Fewer than five team members decrease interaction and results in smaller productivity gains. Having more than nine members requires too much coordination.

The scrum team works together closely, on a daily basis, to ensure the smooth flow of information and the quick resolution of issues. The scrum team delivers product iteratively and incrementally, maximizing opportunities for feedback. Incremental deliveries of a complete product ensure a potentially useful version of working product is always available.

V. Scrum Process Framework

Scrum Process Framework can be viewed by means of a sequence of events and the corresponding artifacts. The Scrum events are time-boxed events. In a project, every scrum event has a predefined maximum duration. These events enable transparency on the project progress to all who are involved in the project. The vital events of scrum are-

- The Sprint
- Sprint Planning
- Daily Scrum Meetings
- The Sprint Review
- The Sprint Retrospective

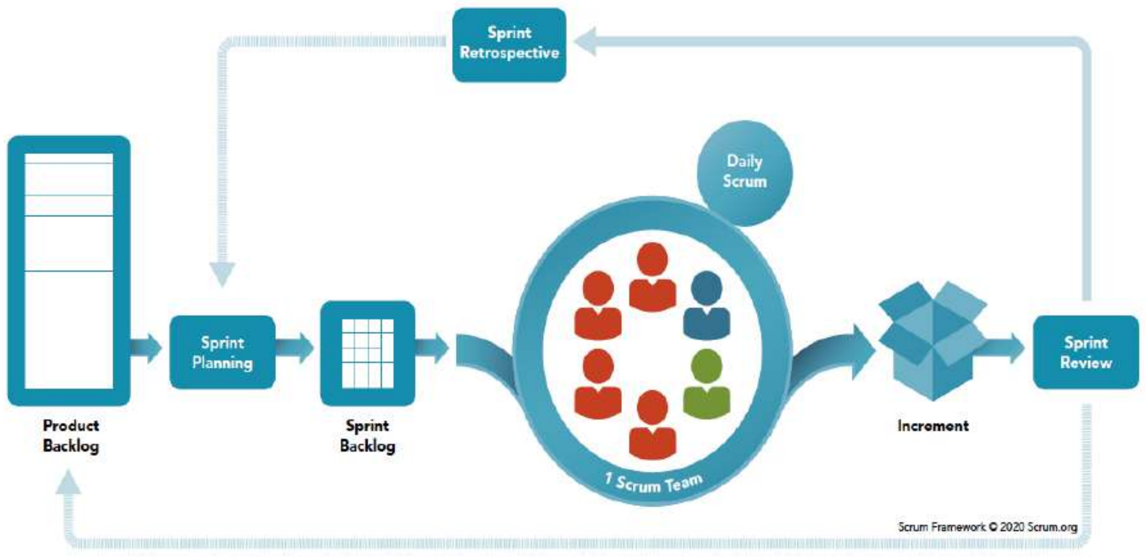


Fig. Scrum Framework

The Sprint

During a Sprint, a working product Increment is developed. It is usually of duration two weeks or one month, and this duration remains constant for all the sprints in the project. A new Sprint starts immediately after the conclusion of the previous Sprint.

The Sprint Goal is an objective set for the Sprint. It provides guidance to the Team on why it is building the Increment. It is created during the Sprint Planning meeting. The scope of the sprint is clarified and re-negotiated between the Product Owner and the Team. Thus, each

Sprint is associated with it, a definition of what is to be built, a design, and the flexible plan that will guide building it, the development work, and the resultant product increment.

A Sprint should be cancelled if the Sprint Goal becomes obsolete. This might occur if the organization changes direction or if market or technology conditions change. A sprint can be cancelled only by product owner, though others have an influence on the same. As the sprint cancellations consume resources, for getting re-organized into another Sprint, they are very uncommon.

If a Sprint is cancelled, and part of the work produced during the sprint is potentially releasable, the Product Owner typically accepts it. All the incomplete Sprint Backlog Items are put back into the Product Backlog.

Sprint Planning

The work to be performed in the Sprint is planned in the Sprint Planning Meeting. Sprint Planning Meeting is of duration of maximum of four hours for two weeks sprints and eight hours for one-month Sprints. It is the responsibility of the Scrum Master to ensure that the meeting takes place and that all the required attendees are present and understand the purpose of the scheduled meeting. The Scrum Master moderates the meeting to monitor the sustenance of discussion and closure on time.

Sprint Planning focuses on the following two questions -

- What needs to be and can be delivered in the Sprint Increment?
- How will the work needed for the execution of Sprint be achieved?

The inputs to this meeting are -

- The Product Backlog
- The latest product Increment
- Projected capacity of the Team during the Sprint
- Past performance of the Team

The Scrum Team discusses the functionality that can be developed during the Sprint. Product Owner provides clarifications on the Product Backlog items. The team selects the items from the Product Backlog for the Sprint, as they are the best to assess what they can accomplish in the Sprint. The Team comprises of analysts, designers, developers, and testers. The work is carried out in a collaborative fashion, thus minimizing re-work.

The Scrum Team then comes up with Sprint Goal. The Sprint Goal is an objective that provides guidance to the Team on why it is building the Product Increment. The Team then decides how it will build the selected functionality into a working product Increment during the Sprint. The Product Backlog items selected for this Sprint plus the plan for delivering them is called the Sprint Backlog.

Work during a sprint is estimated during sprint planning and may be of varying size and/or effort. By the end of the Sprint Planning meeting, the work is divided into tasks of duration of one day or less. This is to enable the ease of work allocation, and tracking the completion. The Team may also invite others (not part of Scrum Team) to attend the Sprint Planning meeting to obtain technical or domain advice or help in estimation.

Daily Scrum Meetings

The Daily Scrum Meeting is a 15-minute meeting for the Team, conducted daily to quickly understand the work since the last Daily Scrum Meeting and create a plan for the next 24 hours. This meeting is also referred to as Daily Stand-up Meeting.

The Daily Scrum Meeting is held at the same time and same place every day to reduce complexity.

During the meeting, each Team member explains -

- What did he do yesterday that helped the Team meet the Sprint Goal?
- What will he do today to help the Team meet the Sprint Goal?
- Does he see any impediments that prevent him or the Team from meeting the Sprint Goal?

Daily Scrum is a planning event. The input to the meeting should be how the team is doing toward meeting the Sprint Goal, and the output should be a new or revised plan that optimizes the team’s efforts in meeting the Sprint Goal.

Though the Scrum Master coordinates the Daily Scrum Meeting and ensures that the objectives of the meeting are met, the Meeting is the responsibility of the Team.

Following are the benefits of Daily Scrum Meetings -

- Improve communication within the Team.
- Identify impediments, if any, in order to facilitate an early removal of the same, so as to minimize impact on the Sprint.

- Highlight and promote quick decision-making.
- Improve the Team's level of knowledge.

Sprint Review

A Sprint Review is held at the end of every Sprint. During the Sprint Review, a presentation of the increment that is getting released is reviewed. In this meeting, the Scrum Team and the stakeholders collaborate to understand what was done in the Sprint. Based on that, and any changes to the Product Backlog during the Sprint, the attendees arrive at the next steps required that could optimize value. Thus, the objective of Sprint Review is to obtain feedback and progress unitedly. The Sprint Review is normally held for two hours for two-week sprints and for four hours for one-month sprints.

The Sprint Review includes the following aspects -

- Attendees include the Scrum Team and key stakeholders, as invited by the Product Owner.
- The Product Owner explains what Product Backlog items have been completed during the sprint and what has not been completed.
- The Team discusses what went well during the Sprint, what problems it ran into, and how those problems were solved.
- The Team demonstrates the work that it has completed and answers questions, if any, about the Increment.
- The entire group then discusses on what to do next. Thus, the Sprint Review provides valuable input to Sprint Planning of the subsequent Sprint.
- The Scrum Team then reviews the timeline, budget, potential capabilities, and marketplace for the next anticipated release of the product increment.
- The outcome of the Sprint Review is an updated Product Backlog, which defines the probable Product Backlog items for the next Sprint.

Sprint Retrospective

The Sprint Retrospective occurs after the Sprint Review and prior to the next Sprint Planning. This is usually a one-hour meeting for two-week duration sprints and a three-hour meeting for one month duration Sprints.

The purpose of the Sprint Retrospective is to -

- Combine the learnings from the last Sprint, with regards to people, relationships, process, and tools.
- Identify the major items that went well and potential improvements.
- Creation of a plan for implementing improvements to increase product quality.

The Sprint Retrospective is an opportunity for the Scrum Team to introspect and improve within the Scrum process framework so as to make the next Sprint outcome more effective.

VI. Scrum Artifacts

Scrum Artifacts provide key information that the Scrum Team and the stakeholders need to be aware of for understanding the product under development, the activities done, and the activities being planned in the project. The following artifacts are defined in Scrum Process Framework -

- Product Backlog
- Sprint Backlog
- Increment

Product Backlog

The Product Backlog is an ordered list of features that are needed as part of the end product and it is the single source of requirements for any changes to be made to the product.

The Product Backlog lists all features, functions, requirements, enhancements, and fixes that constitute the changes to be made to the product in future releases. Product Backlog items have the attributes of a description, order, estimate, and value. These items are normally termed as User Stories. The Product Owner is responsible for the Product Backlog, including its content, availability, and ordering.

A Product Backlog is an evolving artifact. The earliest version of it may contain only the initially known and best understood requirements. The Product Backlog gets developed as the product, and the environment in which it will be used, progress. The Product Backlog constantly changes to incorporate what is required to make it effective. As long as a product exists, its Product Backlog also exists. Product Backlog items can be updated at any time by the Product Owner or at the Product Owner's discretion.

Sprint Backlog

The Sprint Backlog is the set of Product Backlog items selected for the Sprint, plus a plan for delivering the product Increment and realizing the Sprint Goal. The Sprint Backlog will be made available in the next Increment and the work needed to deliver that functionality as a working product Increment.

The Sprint Backlog is a plan with enough detail that can be understood but the Team to track in the Daily Scrum. The Team modifies the Sprint Backlog throughout the Sprint, and the Sprint Backlog emerges during the Sprint. Only the Team can change its Sprint Backlog during a Sprint.

Increment

The Increment is the sum of all the Product Backlog items completed during a Sprint combined with the increments of all previous Sprints. At the end of a Sprint, the new Increment must be a working product. It must be in working condition regardless of whether the Product Owner decides to actually release it.

The Scrum Team members must have a shared understanding of what it means for work to be complete. This is used to assess when work is complete on the product Increment.

The same understanding guides the Team in knowing how many Product Backlog items it can select during a Sprint Planning. The purpose of each Sprint is to deliver Increments of potentially releasable functionality.

Teams deliver an Increment of product functionality every Sprint. This Increment is useable, so a Product Owner may choose to release it immediately. The Scrum Team must define a definition of Increment appropriate for the product. Each Increment is additive to all prior Increments and thoroughly tested, ensuring that all Increments work together.

VII. Advantage of using Scrum framework:

- Scrum framework is fast moving and money efficient.
- Scrum framework works by dividing the large product into small sub-products. It's like a divide and conquer strategy
- In Scrum customer satisfaction is very important.
- Scrum is adaptive in nature because it has short sprint.
- As Scrum framework rely on constant feedback therefore the quality of product increases in less amount of time

VIII. Disadvantage of using Scrum framework:

- Scrum frame work does not allow changes into their sprint.
- Scrum framework is not fully described model. If you want to adopt it you need to fill in the framework with your own details like Extreme Programming (XP), Kanban, DSDM.
- It can be difficult for the Scrum to plan, structure and organize a project that lacks a clear definition. The daily Scrum meetings and frequent reviews require substantial resources.

SOFTWARE ENGINEERING

UNIT – 1

CHAPTER – 4

INTRODUCTION TO DEVOPS AND ITS LIFECYCLE

I. Introduction and benefits of working in a DevOps environment

DevOps and agile are both different models

Similarities

- Both are software development methodologies
- Both models concentrating on rapid software development with team collaboration.

Differences

- The difference will come once development of the project is completed Agile model talks about only development but not operations. DevOps model talks about complete product life cycle like development and operations.
- In Agile model separate people are responsible for development, testing, deployment etc., but in DevOps, the DevOps engineer is responsible for everything like development to operations and operations to development.

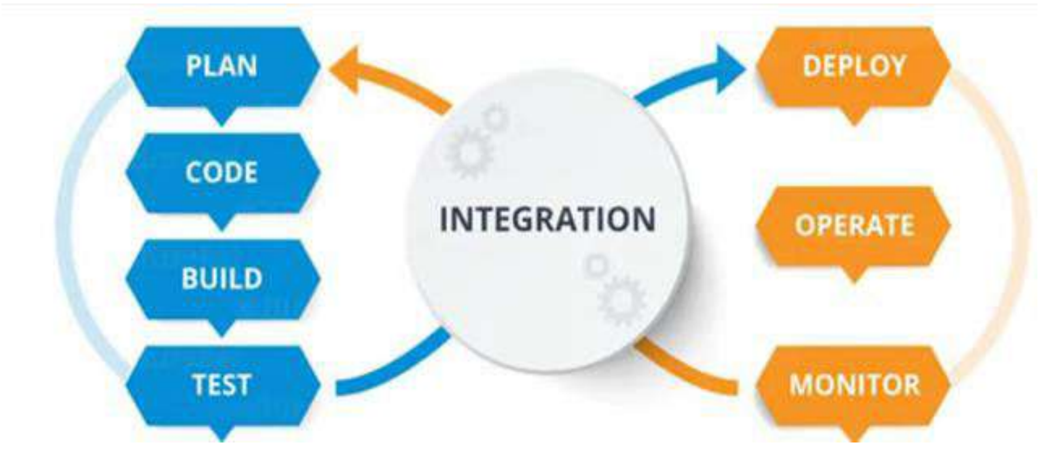
II. What is DevOps

The DevOps is a combination of two words, one is software Development, and second is Operations. This allows a single team to handle the entire application lifecycle, from development to **testing, deployment**, and **operations**. DevOps helps you to reduce the disconnection between software developers, quality assurance (QA) engineers, and system administrators. DevOps promotes collaboration between Development and Operations team to deploy code to production faster in an automated & repeatable way. DevOps helps to increase organization speed to deliver applications and services. It also allows organizations to serve their customers better and compete more strongly in the market.

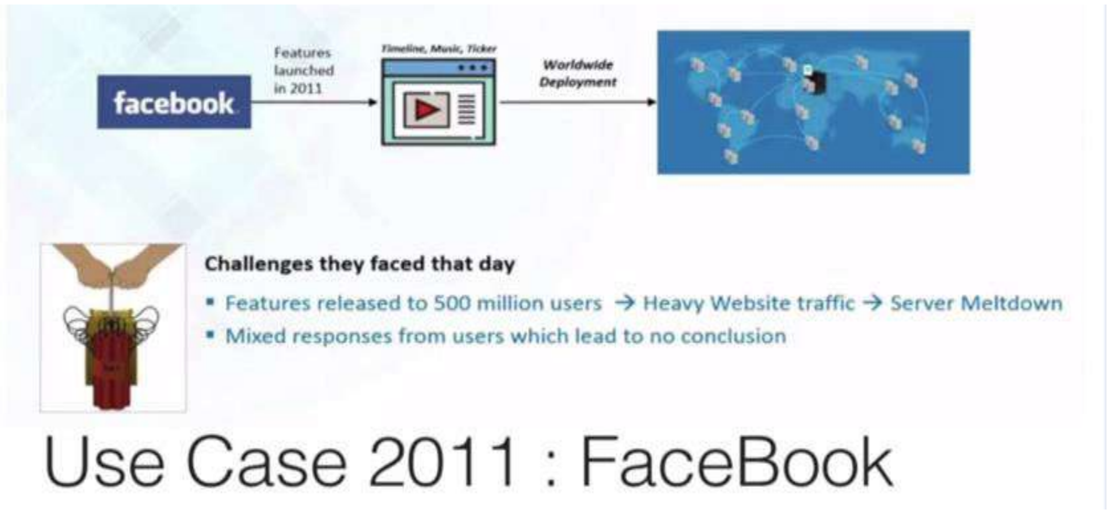
III. Why DevOps?

Before DevOps:

- The operation and development team worked in complete isolation.
- After the design-build, the testing and deployment are performed respectively. That's why they consumed more time than actual build cycles.
- Without the use of DevOps, the team members are spending a large amount of time on designing, testing, and deploying instead of building the project.
- Manual code deployment leads to human errors in production.
- Coding and operation teams have their separate timelines and are not in synch, causing further delays.



IV. USE CASE FOR DevOps


facebook
Features launched in 2011
Timeline, Music, Ticker
Worldwide Deployment

Challenges they faced that day

- Features released to 500 million users → Heavy Website traffic → Server Meltdown
- Mixed responses from users which lead to no conclusion

Use Case 2011 : FaceBook

Facebook released several new features in 2011. One of the most important features was the Timeline. They released the feature worldwide in a single go. They faced some tough challenges. At that time, Facebook had 500 million consumers and all the consumers tried to use the new features which caused the server meltdown as Facebook was not ready for the popularity of their new features. If they would have released the features in multiple releases, then from prior they could have predicted the huge payload and taken prior management so that uninterrupted service can be provided. Also due to this breakdown, Facebook could not get feedback on the features perfectly.

DARK LAUNCHING TECHNIQUE: FACEBOOK

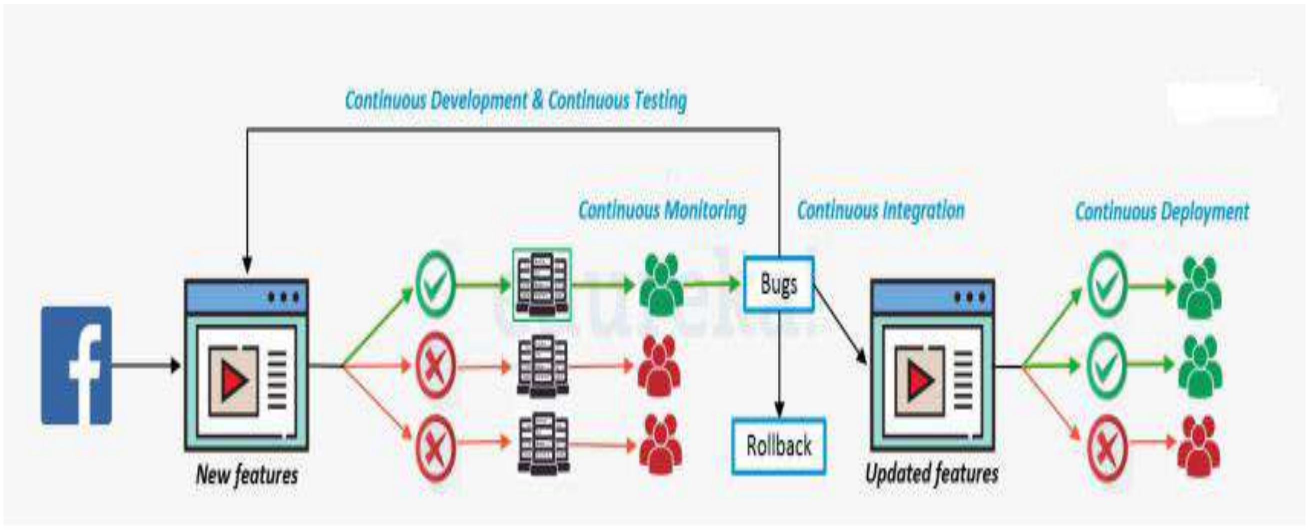


Fig: Facebook comes up with dark launching technique

After this incident of 2011 and the associated challenges, Facebook came up with a new technique named the "**Dark Launching Technique".** Currently, most organizations use this technique for any new release. This technique has three parts.

1. The new features first developed for smaller and specific userbase. Generally, these releases are called beta release or alpha release.
2. They are continuously monitored and feedback is continuously collected and tested.

3. Once the features are stable, they are deployed on the entire userbase in multiple releases

After DevOps

DevOps changed the concept of software development procedure completely. Facebook has one of the richest use-cases of DevOps in the world! Using the DevOps approach, Facebook deploys hundreds and thousands of code updates per day while delivering world-class stability, reliability and security. An early adopter of DevOps, Facebook seamlessly manages bi-weekly app updates that keep two billion users across the world hooked to the app.

By adopting DevOps principles such as continuous deployment, code ownership and incremental release, Facebook is able to get competitive differentiation among its peers. The effectiveness of the process rests on continuous code reviews that dictates that someone has to review and accept the code before the developer can commit it. This ensures better collaboration within and between teams and better perspective on the simpler or scalable path.

By deploying faster and more effectively in more stable operating environments, Facebook has been able to add value rather than just fix or maintain issues. This is why DevOps is highly valued by leading organizations such as Facebook.

A DevOps environment seeks to mitigate such challenges and other issues that may emerge as a result of this model. This is done through various practices, including:

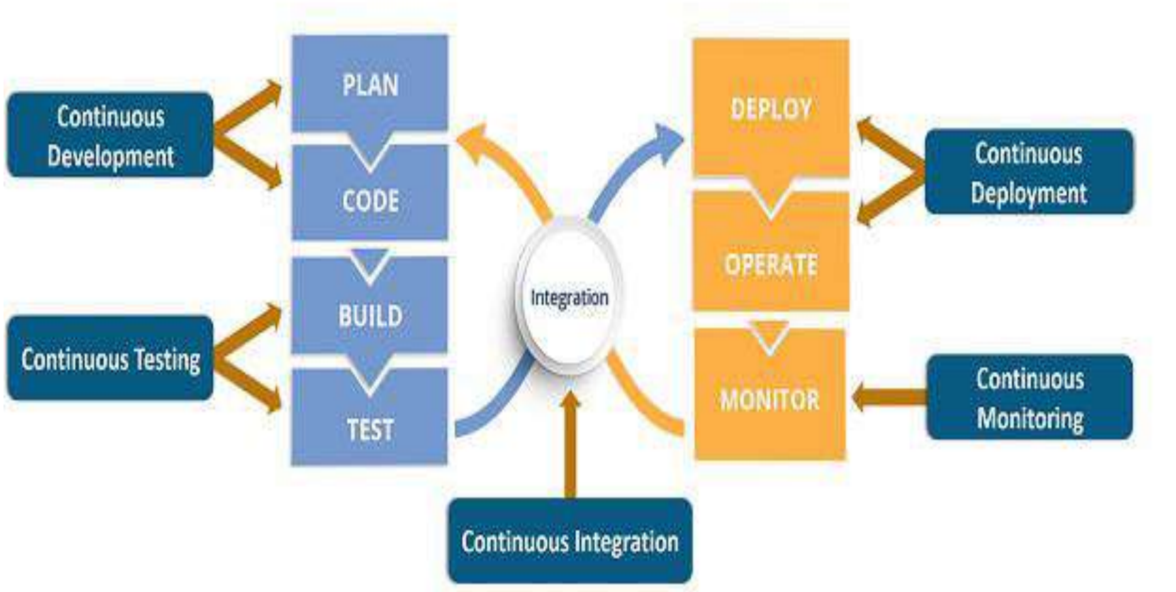
- **Continuous integration:** Integrating code into a shared control repository regularly, preferably daily.
- **Continuous delivery:** Deploying and releasing changes and fixes daily in a controlled, rapid manner.
- **Automated testing:** Testing deployments and deliveries in a continuous, automated fashion.
- **Active monitoring:** Conducting "health-checks" to ensure platforms, applications and solutions are running appropriately.

V. Benefits of working in a DevOps environment

1. Delivering value to customers
2. Reduce cycle time
3. Faster, better product delivery
4. Faster issue resolution and reduced complexity
5. Greater scalability and availability
6. More stable operating environments
7. Better resource utilization
8. Greater automation
9. Greater visibility into system outcomes
10. Greater innovation

LIFECYCLE OF DEVOPS

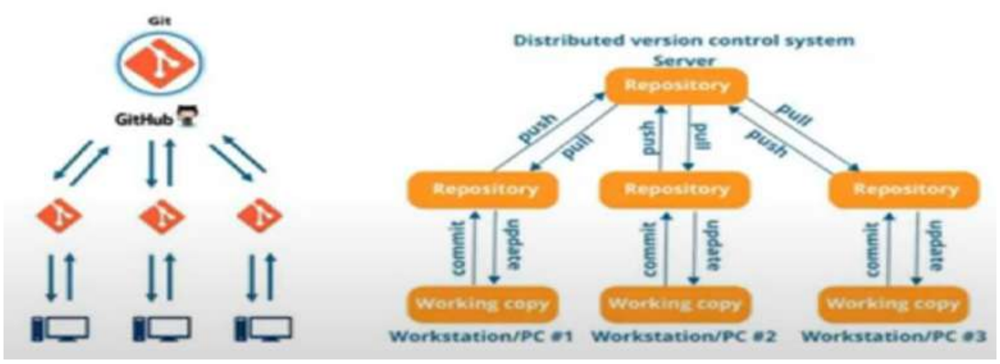
VI. DEVOPS LIFECYCLE



Continuous Development:

- This is the phase which involves planning and coding of the software application functionality. There are no tools for planning as such but there are a number of tools for maintaining the code.

- The vision of the project is decided during the planning phase and then the actual coding of the application begins.
- The code can be written in any language but it is maintained using version control tools these are continuous development tools.
- The tools like git enable communication between the development and the operations team.



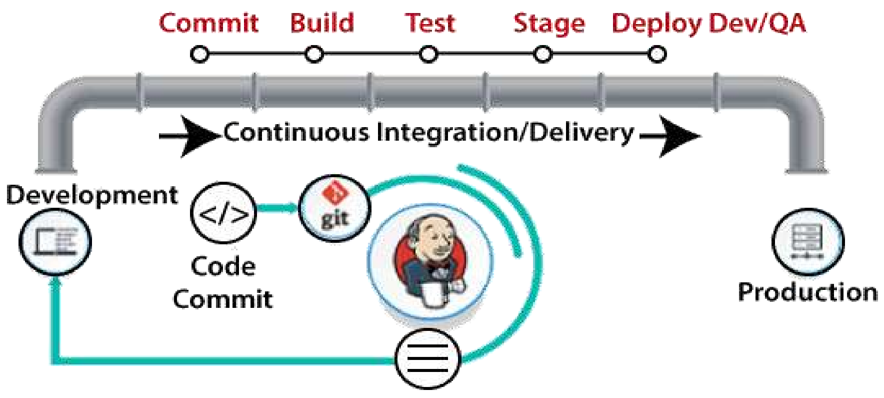
- Git is a distributed version control that supports distributed non-linear workflows by providing data assurance for developing quality software when you are developing a large project with huge number of collaborators
- It is very important to have communication between the collaborators while making the changes in a project so imagine a scenario if you are a team of 10 developers and you are working on a project
- so now what happens is if a developer commits any change into the code and that change causes an error so now how will you track down which developer made what change into code and how to solve that error so here tools such as git is used to solve the problem related to maintain the code
- What happens in git is one central repository or the main server where the code of the application is present and there is also one local repository were you can also have the code for your application
- Pull: you can fetch the code from the main server to local repository using pull
- Push: you can forward the code from the local repository on to the main repository or the main server also this is the working directory or your workspace were you develop the application
- Update: you can fetch the code from local repository onto your working directory using update
- Commit: you can forward your code from working directory to local repository so this an

overview of git

- The advantage of using tools like git: imagine for any reason these server crashes or is unavailable so in such a scenario the local repository still have the code for your application

Continuous Integration:

- This is the stage where the code supporting new functionality is integrated with the existing code since there is continuous development in software
- The updated code needs to be integrated continuously as well as smoothly with the systems to reflect the changes to the end users
- The changed code should also ensure that there are no errors during the runtime which allows us to test the changes and checks how it reacts with other changes so there is one very popular tool that is used in this phase which is known as Jenkins
- Using Jenkins one can pull the latest code version from the git repository and produce a build which can be initially deployed to the test servers or the production servers
- Imagine a developer commits any change onto the code which is on the git repository as soon as there is change in the code on the git repository Jenkins will fetch the code and it will produce a build which is an executable file which is in the form of jar file and this build can be forwarded to the next stages that is either production servers or the test servers.



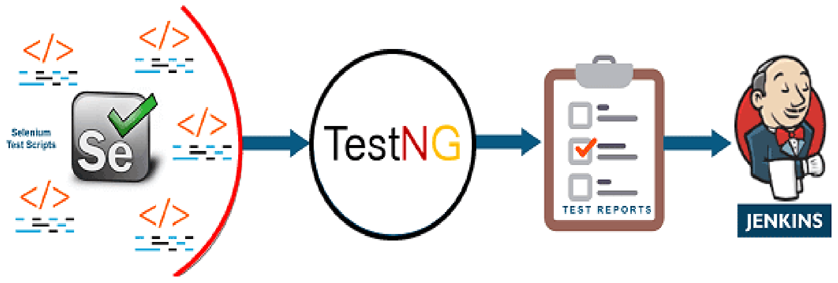
-

Continuous Testing:

- This is the stage were developed software is continuously tested for bugs.
- For continuous Testing automation testing tools such as selenium test ng, JUnit etc are used
- These tools allow the us to test the multiple code bases thoroughly and parallel to ensure that there are no flaws in the functionality
- In this phase you can use docker containers for simulating test environment selenium does

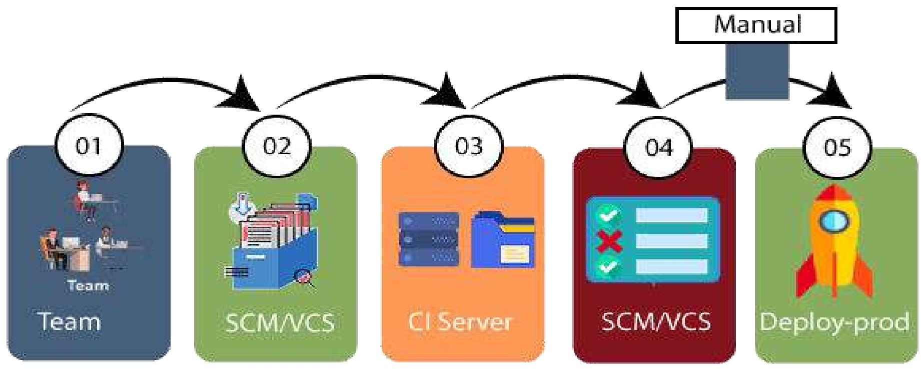
the automated testing and reports are generated by test ng but to automate this entire testing phase you need a trigger and that trigger is provided by the continuous integration tool such as Jenkins

- Automation testing saves a lot of time effort and labour for executing the test cases.
- Besides that, report generation is a big plus the task of evaluating which test cases failed in the test root gets simpler these can also be scheduled for execution at predefined times once the code is tested it is continuously integrated with existing code



Continuous Deployment:

It is the stage where code is deployed to the production environment here, we ensure that code is correctly deployed on all the servers, now it is the time to understand why DevOps will be incomplete without configuration management tools and containerization tools both the set of tools helps us in achieving continuous deployment



Configuration management

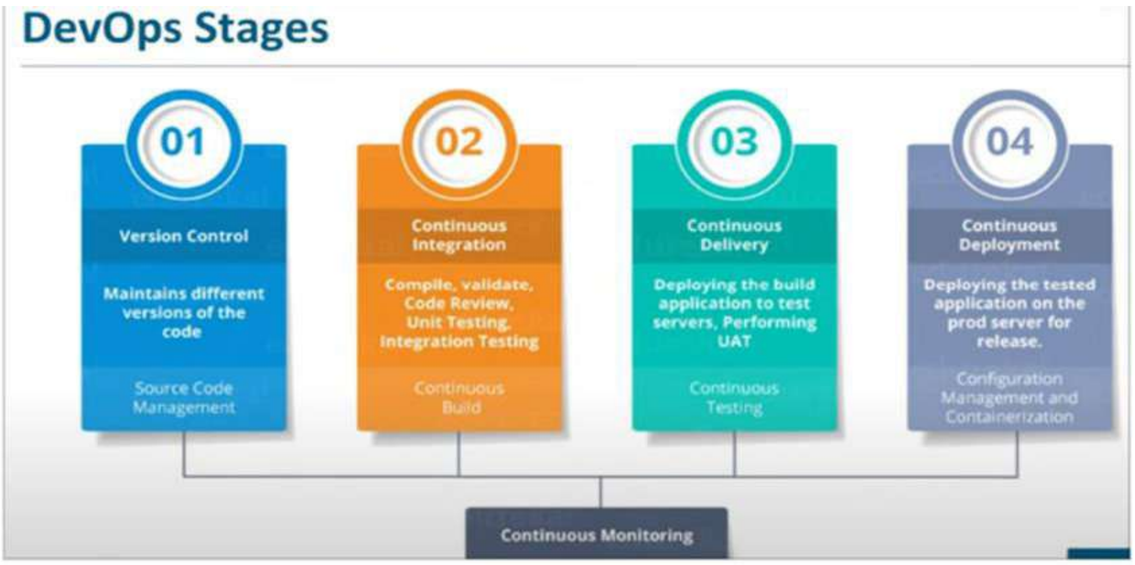
- is the act of establishing and maintaining consistency in an application functional requirements and performance

- It is act of releasing deployments to servers scheduling updates on all the servers and most importantly keeping the configurations consistent across all the servers since the new code is deployed on a continuous basis
- configuration management tools play an important role for executing task quickly and frequently popular tools that are used puppet, chief, Salt stack
- Containerization tools also play equally important role in the deployment stage docker and vagrant are popular tools which helped to reduce consistency across the development test staging and production environment
- Besides this they also help in scaling up and scaling down of instances easily it eliminates any chance of errors or failures in the production environment by packaging and replicating the same dependencies and packages used in development testing and staging environment

Continuous Monitoring

- **T**he final stage in DevOps lifecycle is continuous monitoring this is the crucial stage in DevOps life cycle which is aimed at improving the quality of the software by monitoring its performance.
- This practice involves the participation of the operations team who will monitor the user activity for any bugs or improper behaviour of the system this can also be achieved by making use of dedicated monitoring tools which will continuously monitor the application performance and highlight the issues some popular tools used are Splunk, elk stack, Nagios
- They monitor the application and the servers closely to check the health of the system proactively and they improve productivity and increase the reliability of the system reducing the i.t support costs any major issues found could be reported to the development team so it can be fixed in continuous development phase
- These DevOps stages are carried out on loop continuously until the desired product quality is achieved

VII. DEVOPS STAGES

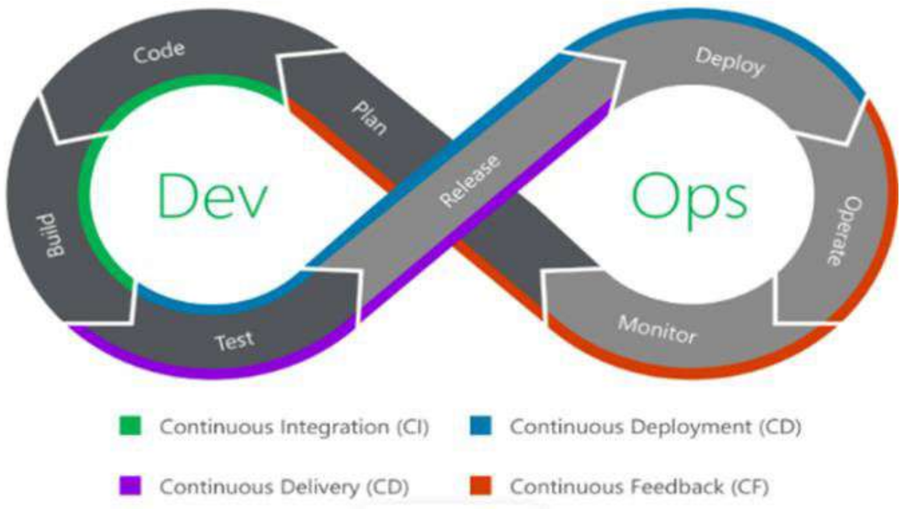

DevOps Stages
01
Version Control
Maintains different versions of the code
Source Code Management
02
Continuous Integration
Compile, validate, Code Review, Unit Testing, Integration Testing
Continuous Build
03
Continuous Delivery
Deploying the build application to test servers, Performing UAT
Continuous Testing
04
Continuous Deployment
Deploying the tested application on the prod server for release.
Configuration Management and Containerization
Continuous Monitoring

The four stages of DevOps are:

1. Version Control: Also called as Source Code Management, used to maintain different versions of the developed source code.
2. Continuous Integration: Also called Continuous Build, where the source code is compiled, validated, reviewed, tested in Unit testing and Integration testing.
3. Continuous Delivery: Also called Continuous Testing, used to deploy the build application to the test servers and perform User Acceptance Test.
4. Continuous Deployment: Also called Configuration Management and Containerization, used to deploy the tested application on the production server for release.

All these four stages are monitored continuously using the continuous monitoring tools like Nagios.

VIII. DEVOPS DELIVERY PIPELINE


Code
Plan
Release
Deploy
Build
Dev
Ops
Operate
Test
Monitor
🟩 Continuous Integration (CI)
🟦 Continuous Deployment (CD)
🟪 Continuous Delivery (CD)
🟥 Continuous Feedback (CF)

The Eight Phases of a DevOps Pipeline

1. Plan

- The Plan stage covers everything that happens before the developers start writing code
- Requirements and feedback are gathered from stakeholders and customers and used to build a product roadmap to guide future development.
- The product roadmap can be recorded and tracked using a ticket management system such as Jira, Azure DevOps or Asana which provide a variety of tools that help track project progress, issues and milestones.
- The product roadmap can be broken down into Epics, Features and User Stories, creating a backlog of tasks that lead directly to the customers' requirements.
- The tasks on the backlog can then be used to plan sprints and allocate tasks to the team to begin development.

2. Code

- In addition to the standard toolkit of a software developer, the team has a standard set of plugins installed in their development environments to aid the development process, help enforce consistent code-styling and avoid common security flaws and code anti-patterns.
- This helps to teach developers good coding practice while aiding collaboration by providing some consistency to the code base. These tools also help resolve issues that may fail tests later in the pipeline, resulting in fewer failed builds.

3. Build

- Once a developer has finished a task, they commit their code to a shared code repository. There are many ways this can be done, but typically the developer submits a pull request — a request to merge their new code with the shared codebase.
- Another developer then reviews the changes they've made, and once they're happy there are no issues, they approve the pull-request. This manual review is supposed to be quick and lightweight, but it's effective at identifying issues early.
- Simultaneously, the pull request triggers an automated process which builds the codebase and runs a series of end-to-end, integration and unit tests to identify any regressions. If the build fails, or any of the tests fail, the pull-request fails and the developer is notified to resolve the issue. By continuously checking code changes into a shared repository and running builds and tests, we can minimise integration issues that arise when working on a shared codebase, and highlight breaking bugs early in the development lifecycle.

4. Test

- Once a build succeeds, it is automatically deployed to a staging environment for deeper, out-of-band testing.
- The staging environment may be an existing hosting service, or it could be a new environment provisioned as part of the deployment process. This practice of automatically provisioning a new environment at the time of deployment is referred to as Infrastructure-as-Code (IaC) and is a core part of many DevOps pipelines.
- Once the application is deployed to the test environment, a series of manual and automated tests are performed. Manual testing can be traditional User Acceptance Testing (UAT) where people use the application as the customer would to highlight any issues or refinements that should be addressed before deploying into production.
- At the same time, automated tests might run security scanning against the application, check for changes to the infrastructure and compliance with hardening best-practices, test the performance of the application or run load testing. The testing that is performed during this phase is up to the organization and what is relevant to the application, but this stage can be considered a test-bed that lets you plug in new testing without interrupting the flow of developers or impacting the production environment.

5. Release

- The Release phase is a milestone in a DevOps pipeline — it's the point at which we say a build is ready for deployment into the production environment. By this stage, each code change has passed a series of manual and automated tests, and the operations team can be confident that breaking issues and regressions are unlikely.

- Depending on the DevOps maturity of an organization, they may choose to automatically deploy any build that makes it to this stage of the pipeline. Developers can use feature flags to turn off new features so they can't be seen by the customers until they are ready for action. This model is considered the nirvana of DevOps and is how organizations manage to deploy multiple releases of their products every day.
- Alternatively, an organization may want to have control over when builds are released to production. They may want to have a regular release schedule or only release new features once a milestone is met. You can add a manual approval process at the release stage which only allows certain people within an organization to authorize a release into production.
- The tooling lets you customize this, it's up to you how you want to go about things.

6. Deploy

- Finally, a build is ready for the big time and it is released into production. There are several tools and processes that can automate the release process to make releases reliable with no outage window.
- The same Infrastructure-as-Code that built the test environment can be configured to build the production environment. We already know that the test environment was built successfully, so we can rest assured that the production release will go off without a hitch.
- A blue-green deployment lets us switch to the new production environment with no outage. Then the new environment is built, it sits alongside the existing production environment. When the new environment is ready, the hosting service points all new requests to the new environment. If at any point, an issue is found with the new build, you can simply tell the hosting service to point requests back to the old environment while you come up with a fix.

7. Operate

- The new release is now live and being used by the customers.
- The operations team is now hard at work, making sure that everything is running smoothly. Based on the configuration of the hosting service, the environment automatically scales with load to handle peaks and troughs in the number of active users.
- The organization has also built a way for their customers to provide feedback on their

service, as well as tooling that helps collect and triage this feedback to help shape the future development of the product. This feedback loop is important — nobody knows what they want more than the customer, and the customer is the world's best testing team, donating many more hours to testing the application than the DevOps pipeline ever could.

8. Monitor

- The 'final' phase of the DevOps cycle is to monitor the environment. this builds on the customer feedback provided in the Operate phase by collecting data and providing analytics on customer behaviour, performance, errors and more.
- We can also do some introspection and monitor the DevOps pipeline itself, monitoring for potential bottlenecks in the pipeline which are causing frustration or impacting the productivity of the development and operations teams.
- All of this information is then fed back to the Product Manager and the development team to close the loop on the process. It would be easy to say this is where the loop starts again, but the reality is that this process is continuous. There is no start or end, just the continuous evolution of a product throughout its lifespan, which only ends when people move on or don't need it any more.