Course Name: Software Engineering
Course Code: CS-505
Course Incharge: Er. Anuj Gupta

Introduction to Software Engineering

Software is a program or set of programs containing instructions that provide desired functionality. And Engineering is the process of designing and building something that serves a particular purpose and finds a cost-effective solution to problems.

Software Engineering *is a systematic, disciplined, quantifiable study and approach to the design, development, operation, and maintenance of a software system.*

Dual Role of Software:

1. As a product –

- It delivers the computing potential across networks of Hardware.
- It enables the Hardware to deliver the expected functionality.
- It acts as an information transformer because it produces, manages, acquires, modifies, displays, or transmits information.

2. As a vehicle for delivering a product –

- It provides system functionality (e.g., payroll system)
- It controls other software (e.g., an operating system)
- It helps build other software (e.g., software tools)

Objectives of Software Engineering:

1. **Maintainability**
   It should be feasible for the software to evolve to meet changing requirements.
2. **Efficiency**
   The software should not make wasteful use of computing devices such as memory, processor cycles, etc.
3. **Correctness**
   A software product is correct if the different requirements as specified in the SRS document have been correctly implemented.
4. **Reusability**
   A software product has good reusability if the different modules of the product can easily be reused to develop new products.
5. **Testability**
   Here software facilitates both the establishment of test criteria and the evaluation of the software with respect to those criteria.
6. **Reliability**
   It is an attribute of software quality. The extent to which a program can be expected to perform its desired function, over an arbitrary time period.

7. **Portability**
   In this case, the software can be transferred from one computer system or environment to another.
8. **Adaptability**
   In this case, the software allows differing system constraints, and the user needs to be satisfied by making changes to the software.
9. **Interoperability** – Capability of 2 or more functional units to process data cooperatively.

Program vs Software Product:

1. A program is a set of instructions that are given to a computer in order to achieve a specific task whereas software is when a program is made available for commercial business and is properly documented along with its licensing.

   Software=Program+documentation+licensing.
2. A program is one of the stages involved in the development of the software, whereas a software development usually follows a life cycle, which involves the feasibility study of the project, requirement gathering, development of a prototype, system design, coding, and testing.

Course Name: Software Engineering
Course Code: CS-505
Course Incharge: Er. Anuj Gupta

What is Software Engineering?

The term **software engineering** is the product of two words, **software**, and **engineering**.

The **software** is a collection of integrated programs.

Software subsists of carefully-organized instructions and code written by developers on any of various particular computer languages.

Computer programs and related documentation such as requirements, design models and user manuals.

Engineering is the application of **scientific** and **practical** knowledge to **invent, design, build, maintain**, and **improve frameworks, processes, etc**.

Software Engineering is an engineering branch related to the evolution of software product using well-defined scientific principles, techniques, and procedures. The result of software engineering is an effective and reliable software product.

Course Name: Software Engineering
Course Code: CS-505
Course Incharge: Er. Anuj Gupta

Why is Software Engineering required?

Software Engineering is required due to the following reasons:

- To manage Large software
- For more Scalability
- Cost Management
- To manage the dynamic nature of software
- For better quality Management

Need of Software Engineering

The necessity of software engineering appears because of a higher rate of progress in user requirements and the environment on which the program is working.

- **Huge Programming:** It is simpler to manufacture a wall than to a house or building, similarly, as the measure of programming become extensive engineering has to step to give it a scientific process.
- **Adaptability:** If the software procedure were not based on scientific and engineering ideas, it would be simpler to re-create new software than to scale an existing one.
- **Cost:** As the hardware industry has demonstrated its skills and huge manufacturing has let down the cost of computer and electronic hardware. But the cost of programming remains high if the proper process is not adapted.
- **Dynamic Nature:** The continually growing and adapting nature of programming hugely depends upon the environment in which the client works. If the quality of the software is continually changing, new upgrades need to be done in the existing one.
- **Quality Management:** Better procedure of software development provides a better and quality software product.

Course Name: Software Engineering
Course Code: CS-505
Course Incharge: Er. Anuj Gupta

Characteristics of a good software engineer

The features that good software engineers should possess are as follows:

Exposure to systematic methods, i.e., familiarity with software engineering principles.

Good technical knowledge of the project range (Domain knowledge).

Good programming abilities.

Good communication skills. These skills comprise of oral, written, and interpersonal skills.

High motivation.

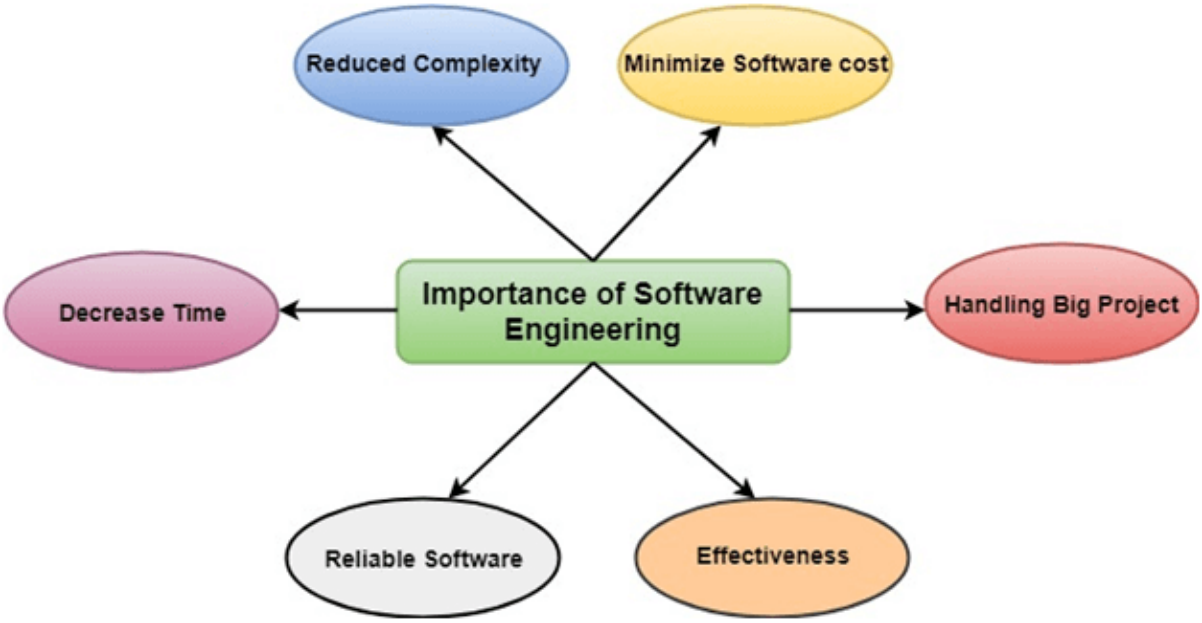
Sound knowledge of fundamentals of computer science.

Intelligence.

Ability to work in a team

Discipline, etc.

Importance of Software Engineering



Course Name: Software Engineering
Course Code: CS-505
Course Incharge: Er. Anuj Gupta

The importance of Software engineering is as follows:

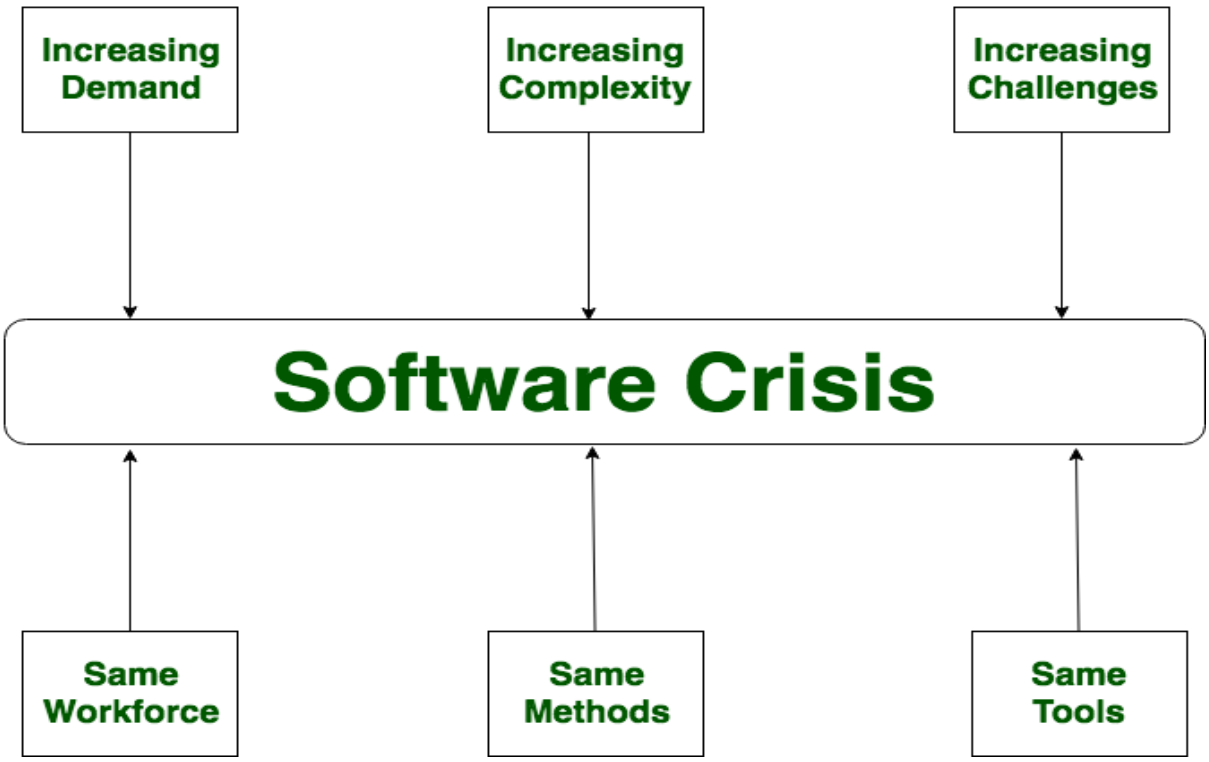
1. **Reduces complexity:** Big software is always complicated and challenging to progress. Software engineering has a great solution to reduce the complication of any project. Software engineering divides big problems into various small issues. And then start solving each small issue one by one. All these small problems are solved independently to each other.
2. **To minimize software cost:** Software needs a lot of hardwork and software engineers are highly paid experts. A lot of manpower is required to develop software with a large number of codes. But in software engineering, programmers project everything and decrease all those things that are not needed. In turn, the cost for software productions becomes less as compared to any software that does not use software engineering method.
3. **To decrease time:** Anything that is not made according to the project always wastes time. And if you are making great software, then you may need to run many codes to get the definitive running code. This is a very time-consuming procedure, and if it is not well handled, then this can take a lot of time. So if you are making your software according to the software engineering method, then it will decrease a lot of time.
4. **Handling big projects:** Big projects are not done in a couple of days, and they need lots of patience, planning, and management. And to invest six and seven months of any company, it requires heaps of planning, direction, testing, and maintenance. No one can say that he has given four months of a company to the task, and the project is still in its first stage. Because the company has provided many resources to the plan and it should be completed. So to handle a big project without any problem, the company has to go for a software engineering method.
5. **Reliable software:** Software should be secure, means if you have delivered the software, then it should work for at least its given time or subscription. And if any bugs come in the software, the company is responsible for solving all these bugs. Because in software engineering, testing and maintenance are given, so there is no worry of its reliability.
6. **Effectiveness:** Effectiveness comes if anything has made according to the standards. Software standards are the big target of companies to make it more effective. So Software becomes more effective in the act with the help of software engineering.

Course Name: Software Engineering
Course Code: CS-505
Course Incharge: Er. Anuj Gupta

Software Crisis

Software Crisis is a term used in computer science for the difficulty of writing useful and efficient computer programs in the required time. software crisis was due to using same workforce, same methods, same tools even though rapidly increasing in software demand, complexity of software and software challenges. With increase in the complexity of software, many software problems arise because existing methods were insufficient.

If we will use same workforce, same methods and same tools after fast increasing in software demand, software complexity and software challenges, then there arise some problems like software budget problem, software efficiency problem, software quality problem, software managing and delivering problem etc. This condition is called software crisis.



Causes of Software Crisis:

- The cost of owning and maintaining software was as expensive as developing the software
- At that time Projects was running over-time
- At that time Software was very inefficient
- The quality of software was low quality
- Software often did not meet requirements
- The average software project overshoots its schedule by half
- At that time Software was never delivered

Solution of Software Crisis:

There is no single solution to the crisis.one possible solution of software crisis is *Software Engineering* because software engineering is a systematic, disciplined and quantifiable approach. For preventing software crisis, there are some guidelines:

- Reduction in software over-budget
- The quality of software must be high
- Less time needed for software project
- Experience working team member on software project
- Software must be delivered

Difference between custom and generic software

Generic software is an off-the-shelf product designed for many consumers and can meet many clients' general requirements. Custom software is a bespoke design developed to meet one client's specific needs, based on the budget and requirements predefined by them. It's meant to be operated by one user or a group of users and meets the needs not fulfilled by off-the-shelf software.
An example of generic software is a word or spreadsheet document. An example of custom software is B2B accounting software or a student portal for a university.

Each of these types has its own pros and cons. Below listed are the main factors that will help you decide what's better.

Unique features

Since generic software is made for a wide range of users, it has plenty of features, all of which may not be useful or needed for a particular organization. GS features may be simple or complex depending on their functionality, but a range of similar GS serving similar use cases will have common features.
On the other hand, custom software will have highly unique and specific features, only limited to the particular client it's made for. Each CS serving a common purpose for different clients will still have distinctly designed functions and features that share no commonality.

Software architecture

GS is super agile and designed to accommodate large-scale future scalability and adaptation as per the growth and pivots in markets and technological advancements.
Custom software may be designed for a specific purpose, considering who is going to use it based on the current business scenario. Still, it may have to incorporate some amount of scalability for the future.

Cost

Generic software is usually not expensive, in that users across demographics can afford it, be they small to medium firms or individual business owners and entrepreneurs. CS is designed to keep in mind the huge strata of the business ecosystem and derives profits from quantity sold and provide decent quality.
Custom software can be expensive since it's developed just for one client. However, if a company invests in custom software to enhance its productivity, efficiency, and customer service and engagement, the returns on software investment are sky-high.

Quality

For generic software producers, creating good quality software is important but not extremely paramount. If a company makes mass consumable software while tinkering with it endlessly until it's perfect beyond comparison, it will lose market share to competitors and fail to make enough profits.
As opposed to generic software, custom-made software is focused on quality as the developers have to ensure that the end result surpasses the client's expectations in terms of performance and useability. Custom software has to offer quality solutions to the client to increase revenues, improve customer service, or even streamline internal business processes. These are precisely the reasons why quality cannot be compromised.

Ownership/control

The software company developing generic software is in control of inception, design, architecture, functions, QA and testing stages of the product. Users can own the software after acquiring the licence to use it.

Custom software is produced and owned fully by the client. The development agency can guide, strategize, and execute on the client's behalf, but the client's final authority and control belong.

Which is better?

There's no better or worse option. It really depends on your requirements and budget.

After the initial investment in customised software, there are lower costs and it pays for itself over time, integrating smoothly with your current set-ups.

If your business has a traditional hierarchical structure, involves record-keeping, personalised communication, inter-departmental updates, and data entry, raising tickets to coordinate with off-site teams, massive marketing campaigns with measurable goals, extensive accounting, you'd be better off with a custom software catering to exact specifications per department and workflows. Custom software also integrates with other software your company is already using.

Readymade software is also prone to hacking whereas custom software can be built to avert hacking and malware. Bespoke software is harder to infiltrate.

Other than that, custom software doesn't have to be a lifetime tool. You can maintain it and utilize it as long as it serves your purpose.

The software development agency that built your software offers technical support and assistance as well. So in case of glitches or breakdown, help is readily available.

On the other hand, readymade software does come with its own set of advantages. You save a lot of initial time and money if you decide to invest in generic software to meet general needs. Over time, with a few workarounds, generic software can keep you in a type of momentum, especially if your business doesn't really depend on excessive note-keeping, data sharing, or constant customer communication.

Another reason to continue using generic software is the time and energy needed in finding the right and trustworthy software development agency to whom you can entrust building your customised software. If you choose a mediocre agency based on lower development quotes and don't do enough due diligence checking out the entirety of their portfolio and do not spend enough time chalking out your complete requirements from the software, the whole project can be delayed, and the result may end up being unsatisfactory. Custom software is meant to solve a specific problem in-house. If your business doesn't need problem-solving software, then generic software could fit you better.

SDLC - Overview

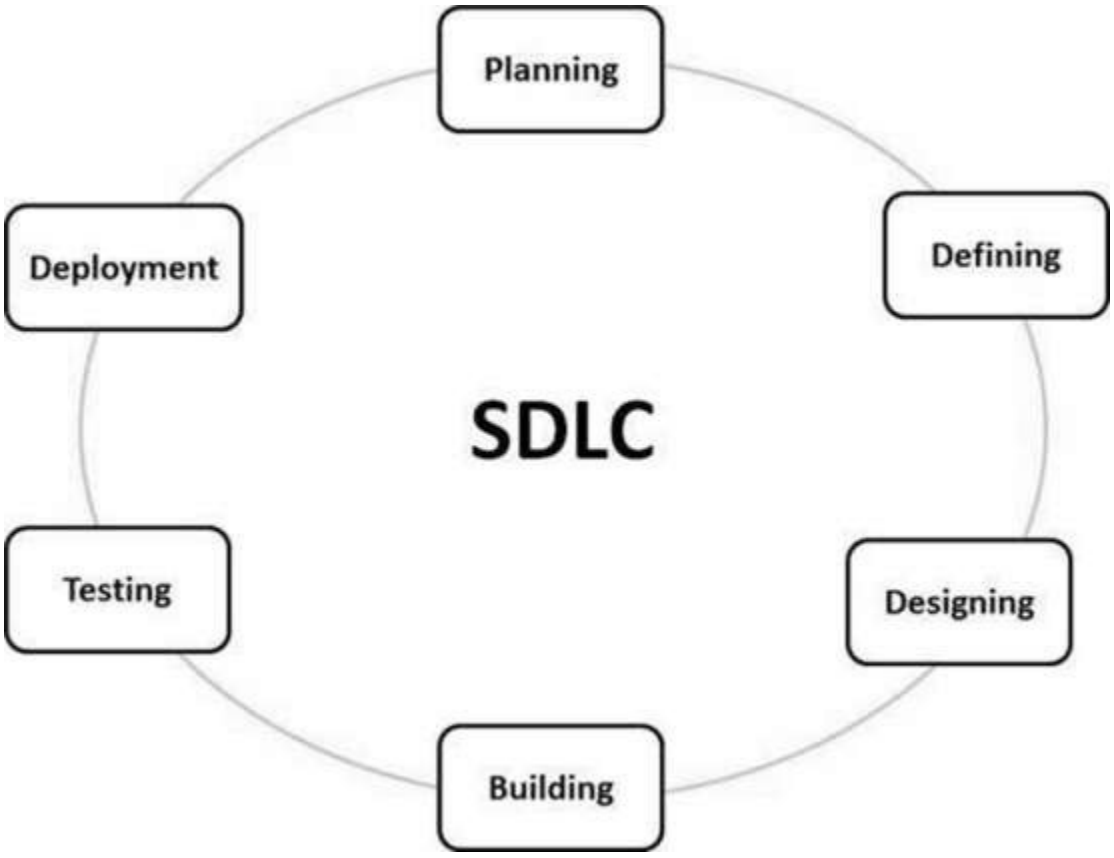
Software Development Life Cycle (SDLC) is a process used by the software industry to design, develop and test high quality softwares. The SDLC aims to produce a high-quality software that meets or exceeds customer expectations, reaches completion within times and cost estimates.

- SDLC is the acronym of Software Development Life Cycle.
- It is also called as Software Development Process.
- SDLC is a framework defining tasks performed at each step in the software development process.
- ISO/IEC 12207 is an international standard for software life-cycle processes. It aims to be the standard that defines all the tasks required for developing and maintaining software.

What is SDLC?

SDLC is a process followed for a software project, within a software organization. It consists of a detailed plan describing how to develop, maintain, replace and alter or enhance specific software. The life cycle defines a methodology for improving the quality of software and the overall development process.

The following figure is a graphical representation of the various stages of a typical SDLC.



A typical Software Development Life Cycle consists of the following stages −

Stage 1: Planning and Requirement Analysis

Requirement analysis is the most important and fundamental stage in SDLC. It is performed by the senior members of the team with inputs from the customer, the sales department, market surveys and domain experts in the industry. This information is then used to plan the basic project approach and to conduct product feasibility study in the economical, operational and technical areas.

Planning for the quality assurance requirements and identification of the risks associated with the project is also done in the planning stage. The outcome of the technical feasibility study is to define the various technical approaches that can be followed to implement the project successfully with minimum risks.

Stage 2: Defining Requirements

Once the requirement analysis is done the next step is to clearly define and document the product requirements and get them approved from the customer or the market analysts. This is done through an **SRS (Software Requirement Specification)** document which consists of all the product requirements to be designed and developed during the project life cycle.

Stage 3: Designing the Product Architecture

SRS is the reference for product architects to come out with the best architecture for the product to be developed. Based on the requirements specified in SRS, usually more than one design approach for the product architecture is proposed and documented in a DDS - Design Document Specification.

This DDS is reviewed by all the important stakeholders and based on various parameters as risk assessment, product robustness, design modularity, budget and time constraints, the best design approach is selected for the product.

A design approach clearly defines all the architectural modules of the product along with its communication and data flow representation with the external and third party modules (if any). The internal design of all the modules of the proposed architecture should be clearly defined with the minutest of the details in DDS.

Stage 4: Building or Developing the Product

In this stage of SDLC the actual development starts and the product is built. The programming code is generated as per DDS during this stage. If the design is performed in a detailed and organized manner, code generation can be accomplished without much hassle.

Developers must follow the coding guidelines defined by their organization and programming tools like compilers, interpreters, debuggers, etc. are used to generate the code. Different high level programming languages such as C, C++, Pascal, Java and PHP are used for coding. The programming language is chosen with respect to the type of software being developed.

Stage 5: Testing the Product

This stage is usually a subset of all the stages as in the modern SDLC models, the testing activities are mostly involved in all the stages of SDLC. However, this stage refers to the testing only stage of the product where product defects are reported, tracked, fixed and retested, until the product reaches the quality standards defined in the SRS.

Stage 6: Deployment in the Market and Maintenance

Once the product is tested and ready to be deployed it is released formally in the appropriate market. Sometimes product deployment happens in stages as per the business strategy of that

organization. The product may first be released in a limited segment and tested in the real business environment (UAT- User acceptance testing).

Then based on the feedback, the product may be released as it is or with suggested enhancements in the targeting market segment. After the product is released in the market, its maintenance is done for the existing customer base.

SDLC Models

There are various software development life cycle models defined and designed which are followed during the software development process. These models are also referred as Software Development Process Models". Each process model follows a Series of steps unique to its type to ensure success in the process of software development.

Following are the most important and popular SDLC models followed in the industry −

- Waterfall Model
- Iterative Model
- Spiral Model
- V-Model
- Big Bang Model

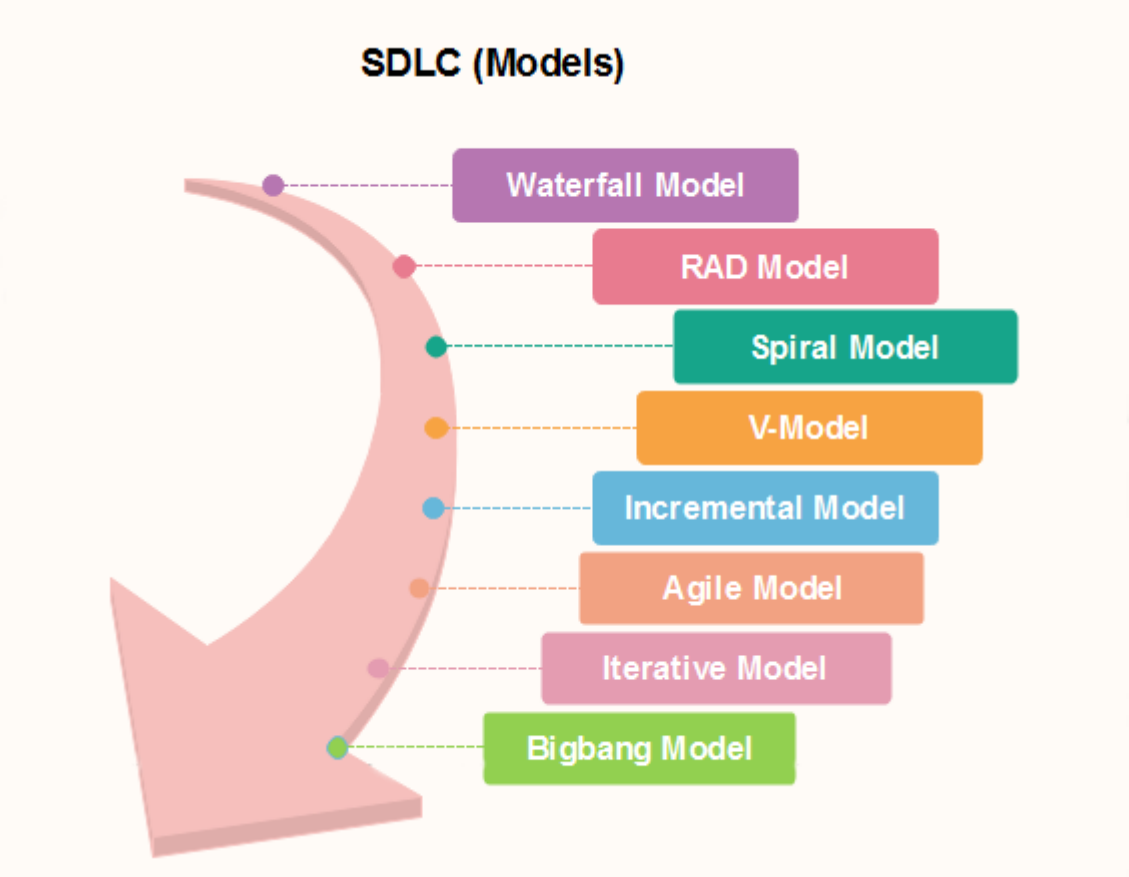
Other related methodologies are Agile Model, RAD Model, Rapid Application Development and Prototyping Models.

SDLC Models

Software Development life cycle (SDLC) is a spiritual model used in project management that defines the stages include in an information system development project, from an initial feasibility study to the maintenance of the completed application.

There are different software development life cycle models specify and design, which are followed during the software development phase. These models are also called "**Software Development Process Models**." Each process model follows a series of phase unique to its type to ensure success in the step of software development.

Here, are some important phases of SDLC life cycle:



Waterfall Model

The waterfall is a universally accepted SDLC model. In this method, the whole process of software development is divided into various phases.

The waterfall model is a continuous software development model in which development is seen as flowing steadily downwards (like a waterfall) through the steps of requirements analysis, design, implementation, testing (validation), integration, and maintenance.

Linear ordering of activities has some significant consequences. First, to identify the end of a phase and the beginning of the next, some certification techniques have to be employed at the end of each step. Some verification and validation usually do this mean that will ensure that the output of the stage is consistent with its input (which is the output of the previous step), and that the output of the stage is consistent with the overall requirements of the system.

RAD Model

RAD or Rapid Application Development process is an adoption of the waterfall model; it targets developing software in a short period. The RAD model is based on the concept that a better system can be developed in lesser time by using focus groups to gather system requirements.

- Business Modeling
- Data Modeling
- Process Modeling
- Application Generation
- Testing and Turnover

Spiral Model

The spiral model is a **risk-driven process model**. This SDLC model helps the group to adopt elements of one or more process models like a waterfall, incremental, etc. The spiral technique is a combination of rapid prototyping and concurrency in design and development activities.

Each cycle in the spiral begins with the identification of objectives for that cycle, the different alternatives that are possible for achieving the goals, and the constraints that exist. This is the first quadrant of the cycle (upper-left quadrant).

The next step in the cycle is to evaluate these different alternatives based on the objectives and constraints. The focus of evaluation in this step is based on the risk perception for the project.

The next step is to develop strategies that solve uncertainties and risks. This step may involve activities such as benchmarking, simulation, and prototyping.

V-Model

In this type of SDLC model testing and the development, the step is planned in parallel. So, there are verification phases on the side and the validation phase on the other side. V-Model joins by Coding phase.

Incremental Model

The incremental model is not a separate model. It is necessarily a series of waterfall cycles. The requirements are divided into groups at the start of the project. For each group, the SDLC model is followed to develop software. The SDLC process is repeated, with each release adding more functionality until all requirements are met. In this method, each cycle act as the maintenance phase for the previous software release. Modification to the incremental model allows development cycles to overlap. After that subsequent cycle may begin before the previous cycle is complete.

Agile Model

Agile methodology is a practice which promotes continues interaction of development and testing during the SDLC process of any project. In the Agile method, the entire project is divided into small incremental builds. All these builds are provided in iterations, and each iteration lasts from one to three weeks.

Any agile software phase is characterized in a manner that addresses several key assumptions about the bulk of software projects:

1. It is difficult to think in advance which software requirements will persist and which will change. It is equally difficult to predict how user priorities will change as the project proceeds.
2. For many types of software, design and development are interleaved. That is, both activities should be performed in tandem so that design models are proven as they are created. It is difficult to think about how much design is necessary before construction is used to test the configuration.
3. Analysis, design, development, and testing are not as predictable (from a planning point of view) as we might like.

Iterative Model

It is a particular implementation of a software development life cycle that focuses on an initial, simplified implementation, which then progressively gains more complexity and a broader feature set until the final system is complete. In short, iterative development is a way of breaking down the software development of a large application into smaller pieces.

Big bang model

Big bang model is focusing on all types of resources in software development and coding, with no or very little planning. The requirements are understood and implemented when they come.

This model works best for small projects with smaller size development team which are working together. It is also useful for academic software development projects. It is an ideal model where requirements are either unknown or final release date is not given.

Prototype Model

The prototyping model starts with the requirements gathering. The developer and the user meet and define the purpose of the software, identify the needs, etc.

A '**quick design**' is then created. This design focuses on those aspects of the software that will be visible to the user. It then leads to the development of a prototype. The customer then checks the prototype, and any modifications or changes that are needed are made to the prototype.

Looping takes place in this step, and better versions of the prototype are created. These are continuously shown to the user so that any new changes can be updated in the prototype. This process continue until the customer is satisfied with the system. Once a user is satisfied, the prototype is converted to the actual system with all considerations for quality and security.

Waterfall Model

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.

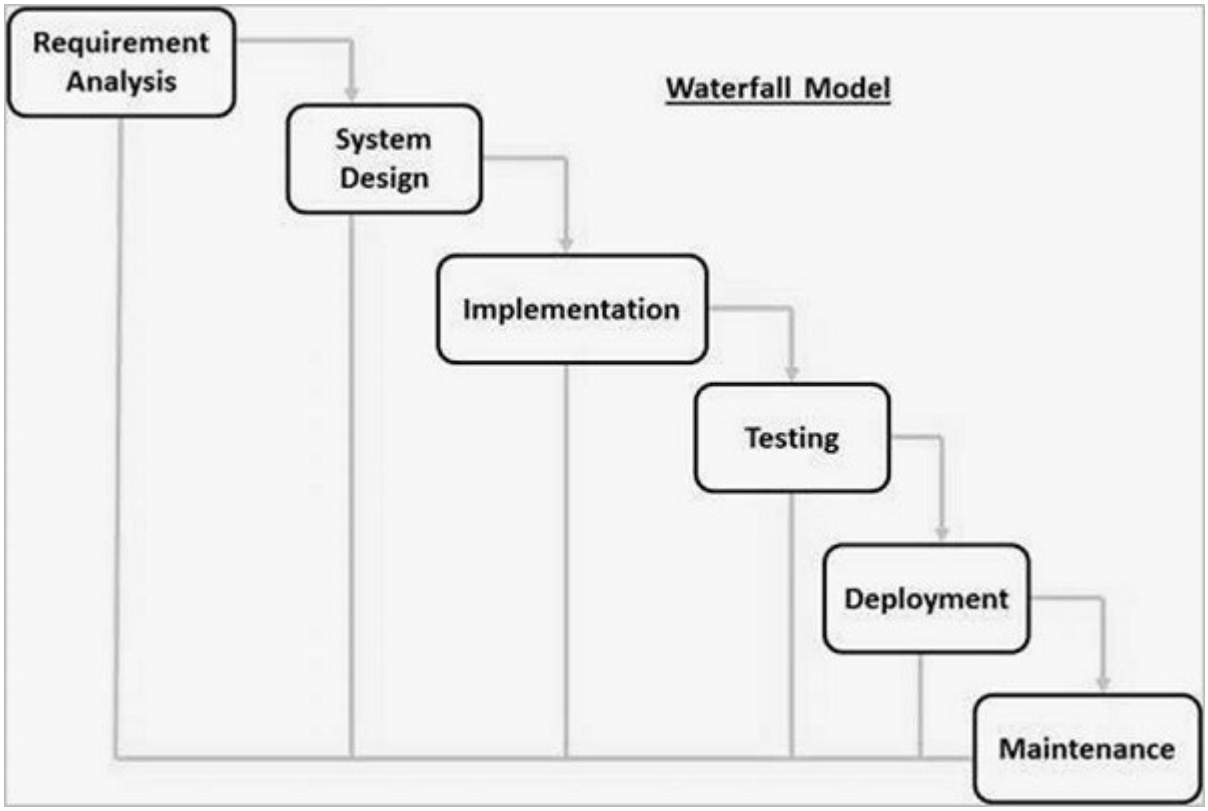
The Waterfall model is the earliest SDLC approach that was used for software development.

The waterfall Model illustrates the software development process in a linear sequential flow. This means that any phase in the development process begins only if the previous phase is complete. In this waterfall model, the phases do not overlap.

Waterfall Model - Design

Waterfall approach was first SDLC Model to be used widely in Software Engineering to ensure success of the project. In "The Waterfall" approach, the whole process of software development is divided into separate phases. In this Waterfall model, typically, the outcome of one phase acts as the input for the next phase sequentially.

The following illustration is a representation of the different phases of the Waterfall Model.



The sequential phases in Waterfall model are −

- **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.

All these phases are cascaded to each other in which progress is seen as flowing steadily downwards (like a waterfall) through the phases. The next phase is started only after the defined set of goals are achieved for previous phase and it is signed off, so the name "Waterfall Model". In this model, phases do not overlap.

Waterfall Model - Application

Every software developed is different and requires a suitable SDLC approach to be followed based on the internal and external factors. Some situations where the use of Waterfall model is most appropriate are −

- Requirements are very well documented, clear and fixed.
- Product definition is stable.
- Technology is understood and is not dynamic.
- There are no ambiguous requirements.
- Ample resources with required expertise are available to support the product.
- The project is short.

Waterfall Model - Advantages

The advantages of waterfall development are that it allows for departmentalization and control. A schedule can be set with deadlines for each stage of development and a product can proceed through the development process model phases one by one.

Development moves from concept, through design, implementation, testing, installation, troubleshooting, and ends up at operation and maintenance. Each phase of development proceeds in strict order.

Some of the major advantages of the Waterfall Model are as follows −

- Simple and easy to understand and use
- Easy to manage due to the rigidity of the model. Each phase has specific deliverables and a review process.
- Phases are processed and completed one at a time.

- Works well for smaller projects where requirements are very well understood.
- Clearly defined stages.
- Well understood milestones.
- Easy to arrange tasks.
- Process and results are well documented.

Waterfall Model - Disadvantages

The disadvantage of waterfall development is that it does not allow much reflection or revision. Once an application is in the testing stage, it is very difficult to go back and change something that was not well-documented or thought upon in the concept stage.

The major disadvantages of the Waterfall Model are as follows −

- No working software is produced until late during the life cycle.
- High amounts of risk and uncertainty.
- Not a good model for complex and object-oriented projects.
- Poor model for long and ongoing projects.
- Not suitable for the projects where requirements are at a moderate to high risk of changing. So, risk and uncertainty is high with this process model.
- It is difficult to measure progress within stages.
- Cannot accommodate changing requirements.
- Adjusting scope during the life cycle can end a project.
- Integration is done as a "big-bang. at the very end, which doesn't allow identifying any technological or business bottleneck or challenges early.

Spiral Model

The spiral model combines the idea of iterative development with the systematic, controlled aspects of the waterfall model. This Spiral model is a combination of iterative development process model and sequential linear development model i.e. the waterfall model with a very high emphasis on risk analysis. It allows incremental releases of the product or incremental refinement through each iteration around the spiral.

Spiral Model - Design

The spiral model has four phases. A software project repeatedly passes through these phases in iterations called Spirals.

Identification

This phase starts with gathering the business requirements in the baseline spiral. In the subsequent spirals as the product matures, identification of system requirements, subsystem requirements and unit requirements are all done in this phase.

This phase also includes understanding the system requirements by continuous communication between the customer and the system analyst. At the end of the spiral, the product is deployed in the identified market.

Design

The Design phase starts with the conceptual design in the baseline spiral and involves architectural design, logical design of modules, physical product design and the final design in the subsequent spirals.

Construct or Build

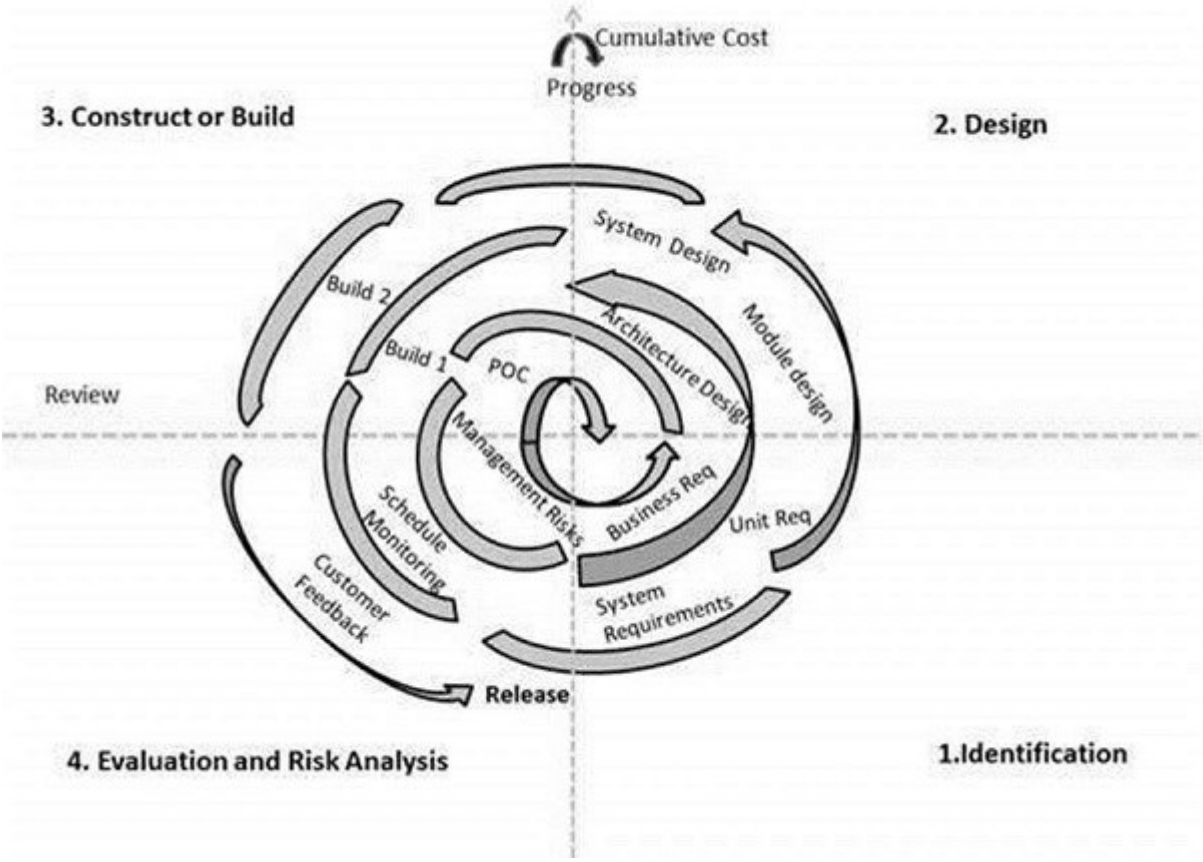
The Construct phase refers to production of the actual software product at every spiral. In the baseline spiral, when the product is just thought of and the design is being developed a POC (Proof of Concept) is developed in this phase to get customer feedback.

Then in the subsequent spirals with higher clarity on requirements and design details a working model of the software called build is produced with a version number. These builds are sent to the customer for feedback.

Evaluation and Risk Analysis

Risk Analysis includes identifying, estimating and monitoring the technical feasibility and management risks, such as schedule slippage and cost overrun. After testing the build, at the end of first iteration, the customer evaluates the software and provides feedback.

The following illustration is a representation of the Spiral Model, listing the activities in each phase.



Based on the customer evaluation, the software development process enters the next iteration and subsequently follows the linear approach to implement the feedback suggested by the customer. The process of iterations along the spiral continues throughout the life of the software.

Spiral Model Application

The Spiral Model is widely used in the software industry as it is in sync with the natural development process of any product, i.e. learning with maturity which involves minimum risk for the customer as well as the development firms.

The following pointers explain the typical uses of a Spiral Model −

- When there is a budget constraint and risk evaluation is important.
- For medium to high-risk projects.
- Long-term project commitment because of potential changes to economic priorities as the requirements change with time.
- Customer is not sure of their requirements which is usually the case.
- Requirements are complex and need evaluation to get clarity.
- New product line which should be released in phases to get enough customer feedback.
- Significant changes are expected in the product during the development cycle.

Spiral Model - Pros and Cons

The advantage of spiral lifecycle model is that it allows elements of the product to be added in, when they become available or known. This assures that there is no conflict with previous requirements and design.

This method is consistent with approaches that have multiple software builds and releases which allows making an orderly transition to a maintenance activity. Another positive aspect of this method is that the spiral model forces an early user involvement in the system development effort.

On the other side, it takes a very strict management to complete such products and there is a risk of running the spiral in an indefinite loop. So, the discipline of change and the extent of taking change requests is very important to develop and deploy the product successfully.

The advantages of the Spiral SDLC Model are as follows −

- Changing requirements can be accommodated.
- Allows extensive use of prototypes.
- Requirements can be captured more accurately.
- Users see the system early.
- Development can be divided into smaller parts and the risky parts can be developed earlier which helps in better risk management.

The disadvantages of the Spiral SDLC Model are as follows −

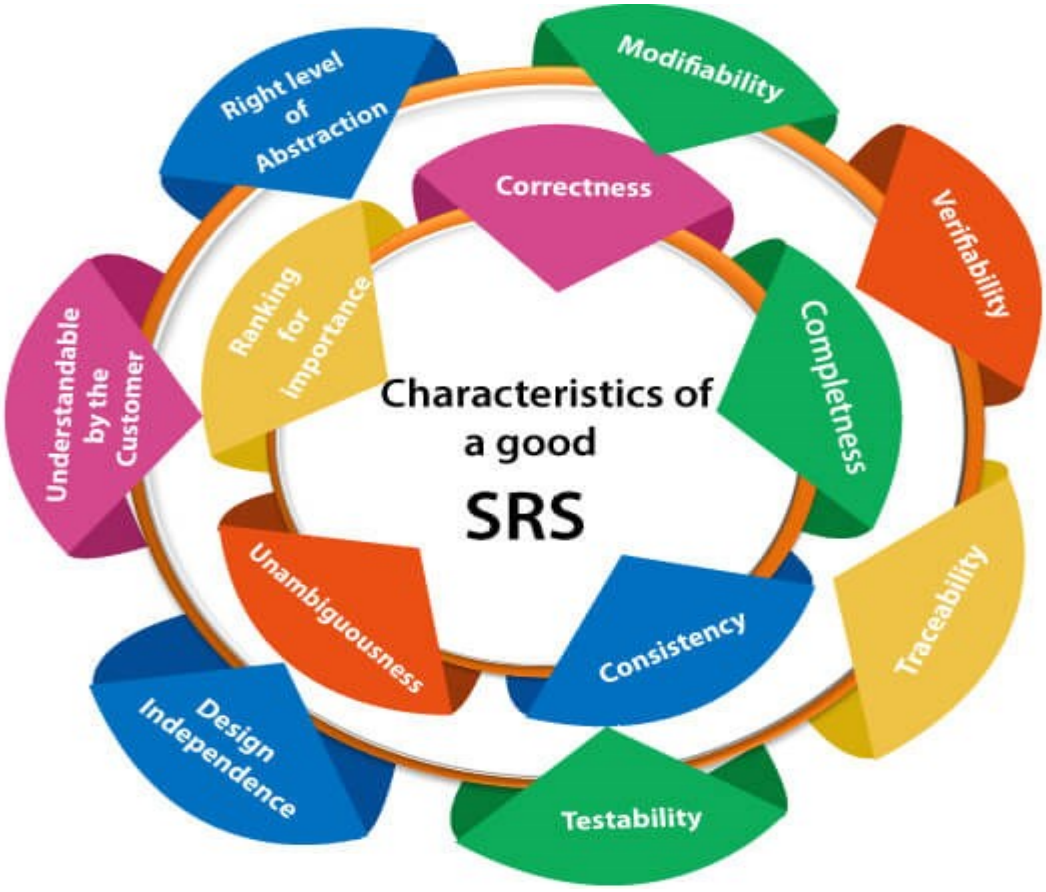
- Management is more complex.
- End of the project may not be known early.
- Not suitable for small or low risk projects and could be expensive for small projects.
- Process is complex
- Spiral may go on indefinitely.
- Large number of intermediate stages requires excessive documentation.

Software Requirement Specifications

The production of the requirements stage of the software development process is **Software Requirements Specifications (SRS)** (also called a **requirements document**). This report lays a foundation for software engineering activities and is constructing when entire requirements are elicited and analysed. **SRS** is a formal report, which acts as a representation of software that enables the customers to review whether it (SRS) is according to their requirements. Also, it comprises user requirements for a system as well as detailed specifications of the system requirements.

The SRS is a specification for a specific software product, program, or set of applications that perform particular functions in a specific environment. It serves several goals depending on who is writing it. First, the SRS could be written by the client of a system. Second, the SRS could be written by a developer of the system. The two methods create entirely various situations and establish different purposes for the document altogether. The first case, SRS, is used to define the needs and expectation of the users. The second case, SRS, is written for various purposes and serves as a contract document between customer and developer.

Characteristics of good SRS



Following are the features of a good SRS document:

1. Correctness: User review is used to provide the accuracy of requirements stated in the SRS. SRS is said to be perfect if it covers all the needs that are truly expected from the system.

2. Completeness: The SRS is complete if, and only if, it includes the following elements:

(1). All essential requirements, whether relating to functionality, performance, design, constraints, attributes, or external interfaces.

(2). Definition of their responses of the software to all realizable classes of input data in all available categories of situations.

(3). Full labels and references to all figures, tables, and diagrams in the SRS and definitions of all terms and units of measure.

3. Consistency: The SRS is consistent if, and only if, no subset of individual requirements described in its conflict. There are three types of possible conflict in the SRS:

(1). The specified characteristics of real-world objects may conflicts. For example,

(a) The format of an output report may be described in one requirement as tabular but in another as textual.

(b) One condition may state that all lights shall be green while another states that all lights shall be blue.

(2). There may be a reasonable or temporal conflict between the two specified actions. For example,

(a) One requirement may determine that the program will add two inputs, and another may determine that the program will multiply them.

(b) One condition may state that "A" must always follow "B," while other requires that "A and B" co-occurs.

(3). Two or more requirements may define the same real-world object but use different terms for that object. For example, a program's request for user input may be called a "prompt" in one requirement's and a "cue" in another. The use of standard terminology and descriptions promotes consistency.

4. Unambiguousness: SRS is unambiguous when every fixed requirement has only one interpretation. This suggests that each element is uniquely interpreted. In case there is a method used with multiple definitions, the requirements report should determine the implications in the SRS so that it is clear and simple to understand.

5. Ranking for importance and stability: The SRS is ranked for importance and stability if each requirement in it has an identifier to indicate either the significance or stability of that particular requirement.

Typically, all requirements are not equally important. Some prerequisites may be essential, especially for life-critical applications, while others may be desirable. Each element should be identified to make these differences clear and explicit. Another way to rank requirements is to distinguish classes of items as essential, conditional, and optional.

6. Modifiability: SRS should be made as modifiable as likely and should be capable of quickly obtain changes to the system to some extent. Modifications should be perfectly indexed and cross-referenced.

7. Verifiability: SRS is correct when the specified requirements can be verified with a cost-effective system to check whether the final software meets those requirements. The requirements are verified with the help of reviews.

8. Traceability: The SRS is traceable if the origin of each of the requirements is clear and if it facilitates the referencing of each condition in future development or enhancement documentation.

There are two types of Traceability:

1. Backward Traceability: This depends upon each requirement explicitly referencing its source in earlier documents.

2. Forward Traceability: This depends upon each element in the SRS having a unique name or reference number.

The forward traceability of the SRS is especially crucial when the software product enters the operation and maintenance phase. As code and design document is modified, it is necessary to be able to ascertain the complete set of requirements that may be concerned by those modifications.

9. Design Independence: There should be an option to select from multiple design alternatives for the final system. More specifically, the SRS should not contain any implementation details.

10. Testability: An SRS should be written in such a method that it is simple to generate test cases and test plans from the report.

11. Understandable by the customer: An end user may be an expert in his/her explicit domain but might not be trained in computer science. Hence, the purpose of formal notations and symbols should be avoided too as much extent as possible. The language should be kept simple and clear.

12. The right level of abstraction: If the SRS is written for the requirements stage, the details should be explained explicitly. Whereas, for a feasibility study, fewer analysis can be used. Hence, the level of abstraction modifies according to the objective of the SRS.

Properties of a good SRS document

The essential properties of a good SRS document are the following:

Concise: The SRS report should be concise and at the same time, unambiguous, consistent, and complete. Verbose and irrelevant descriptions decrease readability and also increase error possibilities.

Structured: It should be well-structured. A well-structured document is simple to understand and modify. In practice, the SRS document undergoes several revisions to cope up with the user requirements. Often, user requirements evolve over a period of time. Therefore, to make the modifications to the SRS document easy, it is vital to make the report well-structured.

Black-box view: It should only define what the system should do and refrain from stating how to do these. This means that the SRS document should define the external behaviour of the system and not discuss the implementation issues. The SRS report should view the system to be developed as a black box and should define the externally visible behaviour of the system. For this reason, the SRS report is also known as the black-box specification of a system.

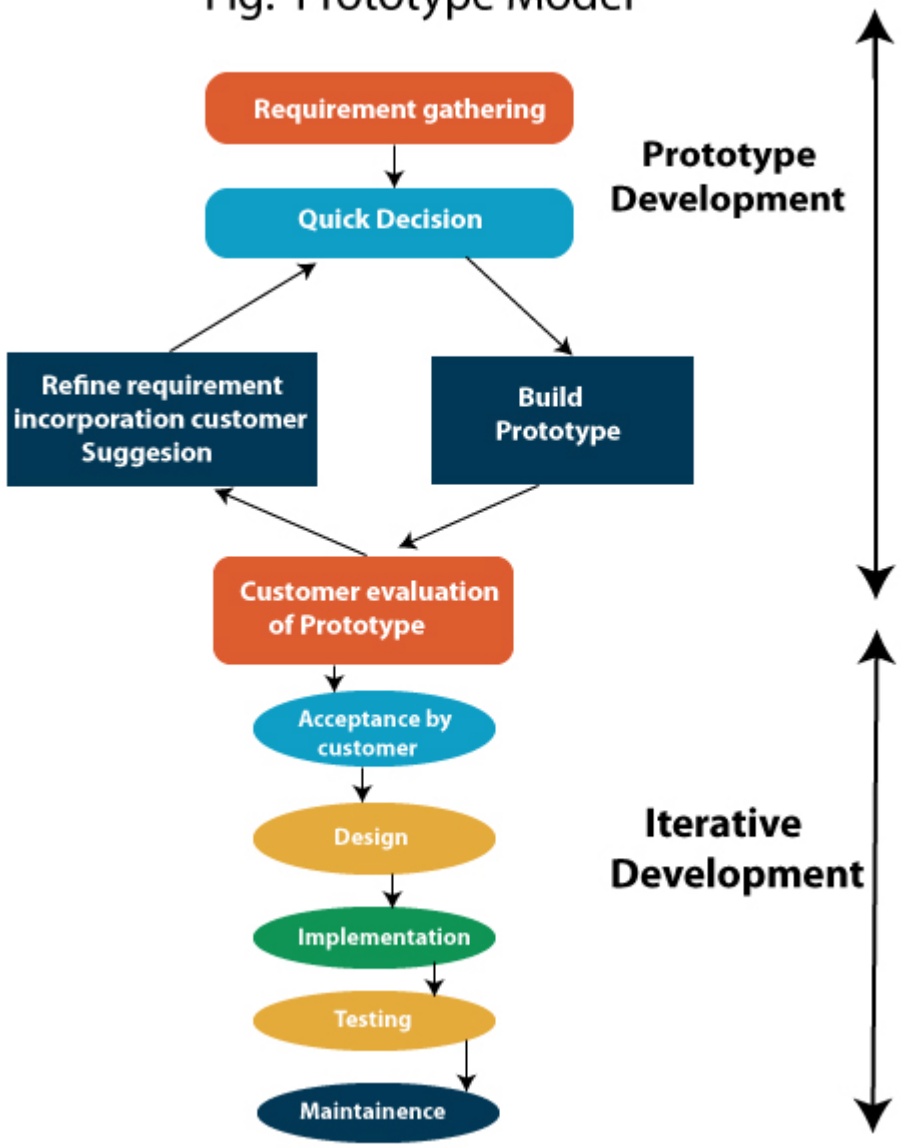
Conceptual integrity: It should show conceptual integrity so that the reader can merely understand it. Response to undesired events: It should characterize acceptable responses to unwanted events. These are called system response to exceptional conditions.

Verifiable: All requirements of the system, as documented in the SRS document, should be correct. This means that it should be possible to decide whether or not requirements have been met in an implementation.

Prototype Model

The prototype model requires that before carrying out the development of actual software, a working prototype of the system should be built. A prototype is a toy implementation of the system. A prototype usually turns out to be a very crude version of the actual system, possible exhibiting limited functional capabilities, low reliability, and inefficient performance as compared to actual software. In many instances, the client only has a general view of what is expected from the software product. In such a scenario where there is an absence of detailed information regarding the input to the system, the processing needs, and the output requirement, the prototyping model may be employed.

Fig: Prototype Model



Steps of Prototype Model

1. Requirement Gathering and Analyst
2. Quick Decision
3. Build a Prototype
4. Assessment or User Evaluation
5. Prototype Refinement
6. Engineer Product

Advantage of Prototype Model

1. Reduce the risk of incorrect user requirement
2. Good where requirement are changing/uncommitted
3. Regular visible process aids management
4. Support early product marketing
5. Reduce Maintenance cost.
6. Errors can be detected much earlier as the system is made side by side.

Disadvantage of Prototype Model

1. An unstable/badly implemented prototype often becomes the final product.
2. Require extensive customer collaboration
    - Costs customer money
    - Needs committed customer
    - Difficult to finish if customer withdraw
    - May be too customer specific, no broad market
3. Difficult to know how long the project will last.
4. Easy to fall back into the code and fix without proper requirement analysis, design, customer evaluation, and feedback.
5. Prototyping tools are expensive.
6. Special tools & techniques are required to build a prototype.
7. It is a time-consuming process.

Evolutionary Process Model

Evolutionary process model resembles the iterative enhancement model. The same phases are defined for the waterfall model occurs here in a cyclical fashion. This model

differs from the iterative enhancement model in the sense that this does not require a useful product at the end of each cycle. In evolutionary development, requirements are implemented by category rather than by priority.

For example, in a simple database application, one cycle might implement the graphical user Interface (GUI), another file manipulation, another queries and another updates. All four cycles must complete before there is a working product available. GUI allows the users to interact with the system, file manipulation allow the data to be saved and retrieved, queries allow user to get out of the system, and updates allows users to put data into the system.

Benefits of Evolutionary Process Model

Use of EVO brings a significant reduction in risk for software projects.

EVO can reduce costs by providing a structured, disciplined avenue for experimentation.

EVO allows the marketing department access to early deliveries, facilitating the development of documentation and demonstration.

Better fit the product to user needs and market requirements.

Manage project risk with the definition of early cycle content.

Uncover key issues early and focus attention appropriately.

Increase the opportunity to hit market windows.

Accelerate sales cycles with early customer exposure.

Increase management visibility of project progress.

Increase product team productivity and motivations.

Software project management

What is Project?

A project is a group of tasks that need to complete to reach a clear result. A project also defines as a set of inputs and outputs which are required to achieve a goal. Projects can vary from simple to difficult and can be operated by one person or a hundred.

Projects usually described and approved by a project manager or team executive. They go beyond their expectations and objects, and it's up to the team to handle logistics and complete the project on time. For good project development, some teams split the project into specific tasks so they can manage responsibility and utilize team strengths.

What is software project management?

Software project management is an art and discipline of planning and supervising software projects. It is a sub-discipline of software project management in which software projects planned, implemented, monitored and controlled.

It is a procedure of managing, allocating and timing resources to develop computer software that fulfills requirements.

In software Project Management, the client and the developers need to know the length, period and cost of the project.

Prerequisite of software project management?

There are three needs for software project management. These are:

1. Time
2. Cost
3. Quality

It is an essential part of the software organization to deliver a quality product, keeping the cost within the client?s budget and deliver the project as per schedule. There are various factors, both external and internal, which may impact this triple factor. Any of three-factor can severely affect the other two.

Project Manager

A project manager is a character who has the overall responsibility for the planning, design, execution, monitoring, controlling and closure of a project. A project manager represents an essential role in the achievement of the projects.

A project manager is a character who is responsible for giving decisions, both large and small projects. The project manager is used to manage the risk and minimize uncertainty. Every decision the project manager makes must directly profit their project.

Role of a Project Manager:

1. Leader

A project manager must lead his team and should provide them direction to make them understand what is expected from all of them.

2. Medium:

The Project manager is a medium between his clients and his team. He must coordinate and transfer all the appropriate information from the clients to his team and report to the senior management.

3. Mentor:

He should be there to guide his team at each step and make sure that the team has an attachment. He provides a recommendation to his team and points them in the right direction.

Responsibilities of a Project Manager:

1. Managing risks and issues.
2. Create the project team and assigns tasks to several team members.
3. Activity planning and sequencing.
4. Monitoring and reporting progress.
5. Modifies the project plan to deal with the situation.

Activities

Software Project Management consists of many activities, that includes planning of the project, deciding the scope of product, estimation of cost in different terms, scheduling of tasks, etc.

The list of activities are as follows:

1. Project planning and Tracking
2. Project Resource Management
3. Scope Management
4. Estimation Management
5. Project Risk Management
6. Scheduling Management
7. Project Communication Management
8. Configuration Management

Now we will discuss all these activities -

1. Project Planning: It is a set of multiple processes, or we can say that it a task that performed before the construction of the product starts.

2. Scope Management: It describes the scope of the project. Scope management is important because it clearly defines what would do and what would not. Scope Management create the

project to contain restricted and quantitative tasks, which may merely be documented and successively avoids price and time overrun.

3. Estimation management: This is not only about cost estimation because whenever we start to develop software, but we also figure out their size(line of code), efforts, time as well as cost.

If we talk about the size, then Line of code depends upon user or software requirement.

If we talk about effort, we should know about the size of the software, because based on the size we can quickly estimate how big team required to produce the software.

If we talk about time, when size and efforts are estimated, the time required to develop the software can easily determine.

And if we talk about cost, it includes all the elements such as:

- Size of software
- Quality
- Hardware
- Communication
- Training
- Additional Software and tools
- Skilled manpower

4. Scheduling Management: Scheduling Management in software refers to all the activities to complete in the specified order and within time slotted to each activity. Project managers define multiple tasks and arrange them keeping various factors in mind.

For scheduling, it is compulsory -

- Find out multiple tasks and correlate them.
- Divide time into units.
- Assign the respective number of work-units for every job.
- Calculate the total time from start to finish.
- Break down the project into modules.

5. Project Resource Management: In software Development, all the elements are referred to as resources for the project. It can be a human resource, productive tools, and libraries.

Resource management includes:

- Create a project team and assign responsibilities to every team member
- Developing a resource plan is derived from the project plan.
- Adjustment of resources.

6. Project Risk Management: Risk management consists of all the activities like identification, analyzing and preparing the plan for predictable and unpredictable risk in the project.

Several points show the risks in the project:

- The Experienced team leaves the project, and the new team joins it.
- Changes in requirement.
- Change in technologies and the environment.
- Market competition.

7. Project Communication Management: Communication is an essential factor in the success of the project. It is a bridge between client, organization, team members and as well as other stakeholders of the project such as hardware suppliers.

From the planning to closure, communication plays a vital role. In all the phases, communication must be clear and understood. Miscommunication can create a big blunder in the project.

8. Project Configuration Management: Configuration management is about to control the changes in software like requirements, design, and development of the product.

The Primary goal is to increase productivity with fewer errors.

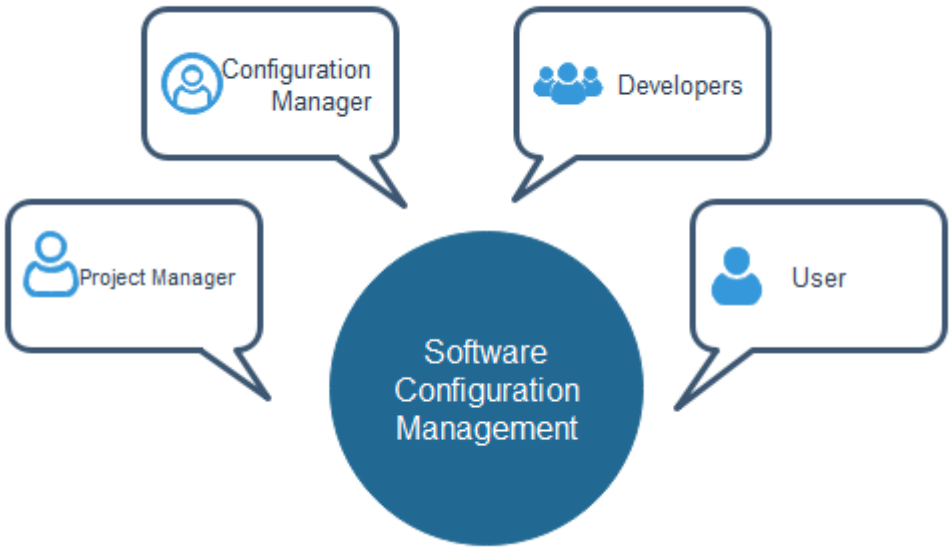
Some reasons show the need for configuration management:

- Several people work on software that is continually update.
- Help to build coordination among suppliers.
- Changes in requirement, budget, schedule need to accommodate.
- Software should run on multiple systems.

Tasks perform in Configuration management:

- Identification
- Baseline
- Change Control
- Configuration Status Accounting
- Configuration Audits and Reviews

People involved in Configuration Management:



Project Management Tools

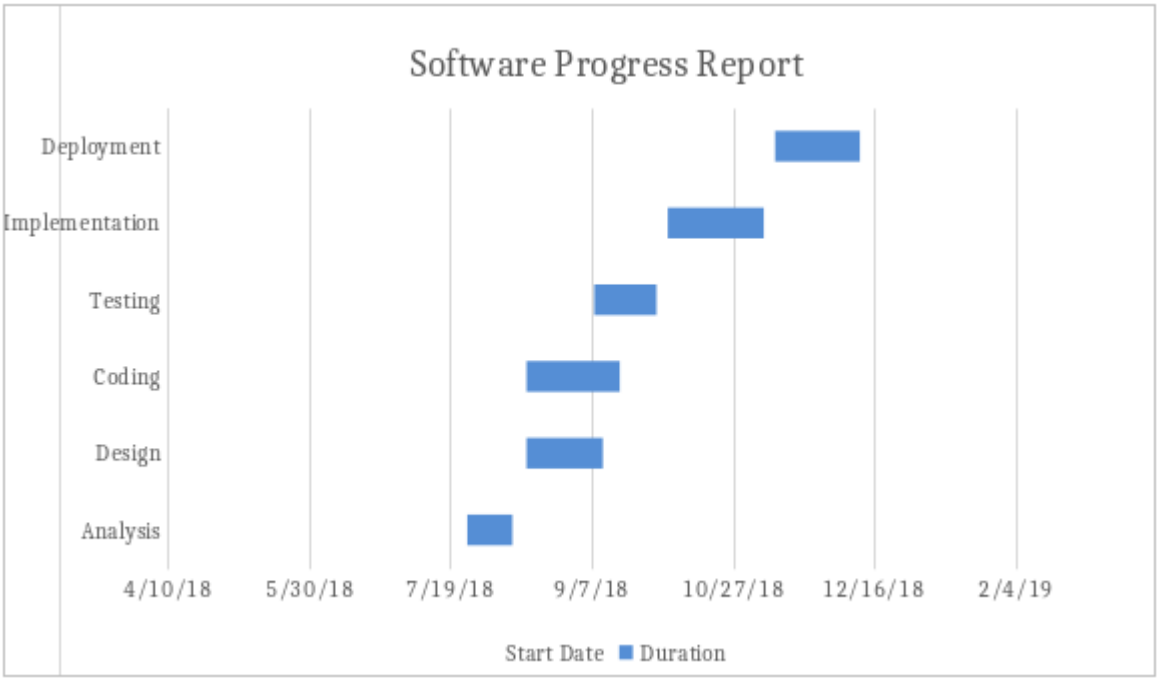
To manage the Project management system adequately and efficiently, we use Project management tools.

Here are some standard tools:

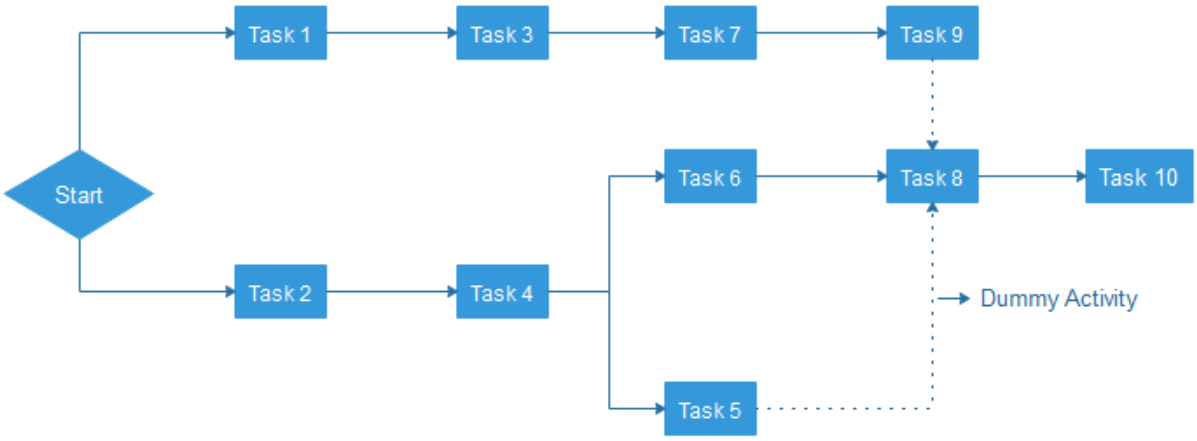
Gantt chart

Gantt Chart first developed by Henry Gantt in 1917. Gantt chart usually utilized in project management, and it is one of the most popular and helpful ways of showing activities displayed against time. Each activity represented by a bar.

Gantt chart is a useful tool when you want to see the entire landscape of either one or multiple projects. It helps you to view which tasks are dependent on one another and which event is coming up.



PERT chart



PERT is an acronym of Programme Evaluation Review Technique. In the 1950s, it is developed by the U.S. Navy to handle the Polaris submarine missile programme.

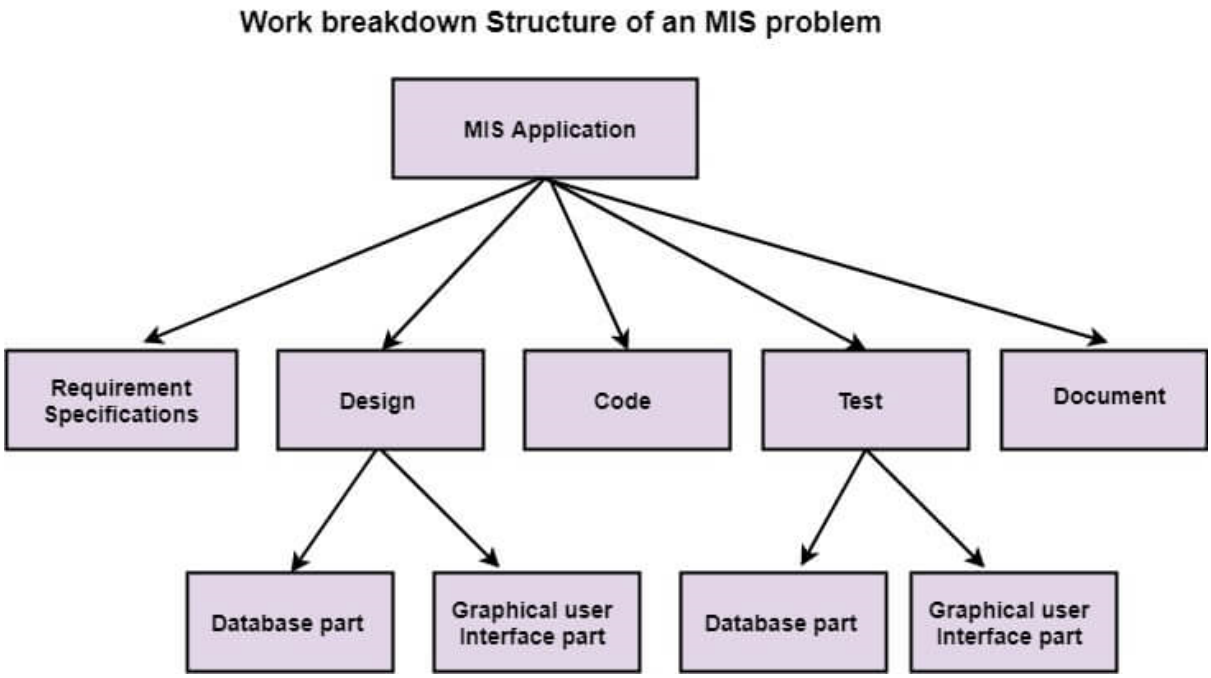
In Project Management, PERT chart represented as a network diagram concerning the number of nodes, which represents events.

The direction of the lines indicates the sequence of the task. In the above example, tasks between "Task 1 to Task 9" must complete, and these are known as a dependent or serial task. Between Task 4 and 5, and Task 4 and 6, nodes are not depended and can undertake simultaneously. These are known as Parallel or concurrent tasks. Without resource or completion time, the task must complete in the sequence which is considered as event dependency, and these are known as Dummy activity and represented by dotted lines.

Logic Network

The Logic Network shows the order of activities over time. It shows the sequence in which activities are to do. Distinguishing events and pinning down the project are the two primary uses. Moreover, it will help with understanding task dependencies, a timescale, and overall project workflow.

Product Breakdown Structure



Product Breakdown Structure (BBS) is a management tool and necessary a part of the project designing. It's a task-oriented system for subdividing a project into product parts. The product breakdown structure describes subtasks or work packages and represents the connection between work packages. Within the product breakdown Structure, the project work has diagrammatically pictured with various types of lists. The product breakdown structure is just like the work breakdown structure (WBS).

Work Breakdown Structure

It is an important project deliverable that classifies the team's work into flexible segments. "Project Management Body of Knowledge (PMBOK)" is a group of terminology that describes the work breakdown structure as a "deliverable-oriented hierarchical breakdown of the work which is performed by the project team."

There are two ways to generate a Work Breakdown Structure ? The top-down and

The bottom-up approach.

In the **top-down approach**, the WBS derived by crumbling the overall project into subprojects or lower-level tasks.

The **bottom-up approach** is more alike to a brainstorming exercise where team members are asked to make a list of low-level tasks which is required to complete the project.

Resource Histogram

The resource histogram is precisely a bar chart that used for displaying the amounts of time that a resource is scheduled to be worked on over a prearranged and specific period. Resource histograms can also contain the related feature of resource availability, used for comparison on purposes of contrast.

Critical Path Analysis

Critical path analysis is a technique that is used to categorize the activities which are required to complete a task, as well as classifying the time which is needed to finish each activity and the relationships between the activities. It is also called a critical path method. CPA helps in predicting whether a project will expire on time.

Software Quality Management

Quality dimension:

David Garvin suggests that quality ought to be thought-about by taking a third-dimensional read point that begins with an assessment of correspondence and terminates with a transcendental (aesthetic) view. Though Garvin's 8 dimensions of quality weren't developed specifically for the software system, they'll be applied once software system quality is taken into account.

Eight dimensions of product quality management will be used at a strategic level to investigate quality characteristics. The idea was outlined by David A. Garvin, formerly C. Roland Christensen academician of Business Administration at Harvard grad school (died thirty Gregorian calendar month 2017).

A number of the scale are reciprocally reinforcing, whereas others don't seem to be, improvement in one is also at the expense of others. Understanding the trade-offs desired by customers among these dimensions will facilitate build a competitive advantage.

Garvin's eight dimensions will be summarized as follows:

1. Performance Quality:

Will the software system deliver all content, functions, and options that are such as a part of the necessities model during a method that gives worth to the tip user?

2. Feature Quality:

Does the software system offer options that surprise and delight first-time finish users?

3. Reliability:

Will the software system deliver all options and capability while not failure?

Is it obtainable once it's needed?

Will it deliver practicality that's error-free?

4. Conformance:

Will the software system adjust to native and external software standards that are relevant to the application?

Will it conform to the factual style and writing conventions? as an example, will the computer program conform to accepted style rules for menu choice or knowledge input?

5. Durability:

Will the software system be maintained (changed) or corrected (debugged) while not the accidental generation of unintentional facet effects? can changes cause the error rate or responsibility to degrade with time?

6. Serviceability:

Will the software system be maintained (changed) or corrected (debugged) in a tolerably short time period?

Will support employees acquire all data they have to create changes or correct defects?

Stephen A. Douglas Adams makes a wry comment that appears acceptable here: "The distinction between one thing that may get it wrong and something that can't probably go wrong is that once something that can't possibly go wrong goes wrong it always seems to be not possible to urge at or repair."

7. Aesthetics:

There's no doubt that every folk includes a totally different and really subjective vision of what's aesthetic.

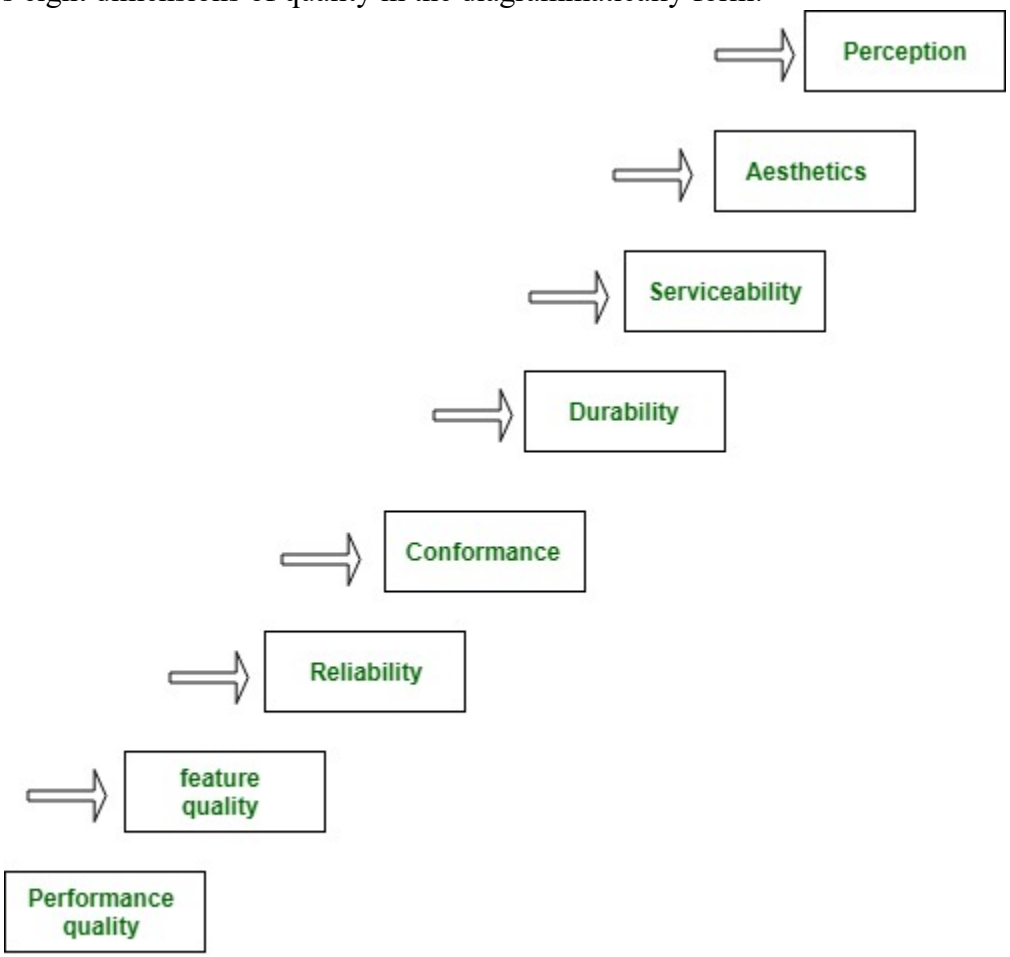
And yet, most folks would agree that an aesthetic entity includes a sure class, a novel flow, and a clear "presence" that are arduous to quantify however are evident still. The aesthetic software system has these characteristics.

8. Perception:

In some things, you've got a collection of prejudices which will influence your perception of quality. as an example, if you're introduced to a software product that was engineered by a seller United Nations agency has created poor quality within the past, your guard is raised and your perception of the present software product quality may be influenced negatively.

Similarly, if a seller has a wonderful name, you will understand quality, even once it doesn't very exist.

Garvin's eight dimensions of quality in the diagrammatically form:



Garvin's Dimentions of Quality

Garvin's quality dimensions offer you with a "soft" take a look at software system quality. several (but not all) of those dimensions will solely be thought-about subjectively. For this reason, you furthermore might want a collection of "hard" quality factors that may be classified in 2 broad groups:

- Factors that can be directly measured (e.g., defects uncovered during testing).
- Factors that may be measured solely indirectly (e.g., usability or maintainability).

In every case, activity should occur. you ought to compare the software system to some information and attain a sign of quality.

Process quality and Product quality (See separate PDF uploaded in google classroom for details):

Product Standards:

- Design review form
- Document naming standards
- Function prototype format
- Programming style standards
- Project plan format
- Change request form

Process Standards:

- Design review guidelines
- Document submission procedures
- Version release process
- Project plan approval procedure
- Change control process
- Test data recording procedures

Quality assurance planning, Quality measurement, software configuration management, software process improvement, ISO 9000 quality standards, ISO approach to quality assurance systems, SEI capability maturity model (CMM), PSP:

What is Quality?

Quality is extremely hard to define, and it is simply stated: "Fit for use or purpose." It is all about meeting the needs and expectations of customers with respect to functionality, design, reliability, durability, & price of the product.

What is Assurance?

Assurance is nothing but a positive declaration on a product or service, which gives confidence. It is certainty of a product or a service, which it will work well. It provides a guarantee that the product will work without any problems as per the expectations or requirements.

Quality Assurance in Software Testing

Quality Assurance in Software Testing is defined as a procedure to ensure the quality of software products or services provided to the customers by an organization. Quality assurance focuses on improving the software development process and making it efficient and effective as per the quality standards defined for software products. Quality Assurance is popularly known as QA Testing.

How to do Quality Assurance: Complete Process

Quality Assurance methodology has a defined cycle called PDCA cycle or Deming cycle. The phases of this cycle are:

- Plan
- Do

- Check
- Act

Quality Assurance Process

These above steps are repeated to ensure that processes followed in the organization are evaluated and improved on a periodic basis. Let's look into the above QA Process steps in detail –

- Plan – Organization should plan and establish the process related objectives and determine the processes that are required to deliver a high-Quality end product.
- Do – Development and testing of Processes and also "do" changes in the processes
- Check – Monitoring of processes, modify the processes, and check whether it meets the predetermined objectives
- Act – A Quality Assurance tester should implement actions that are necessary to achieve improvements in the processes

An organization must use Quality Assurance to ensure that the product is designed and implemented with correct procedures. This helps reduce problems and errors, in the final product.

What is Quality Control?

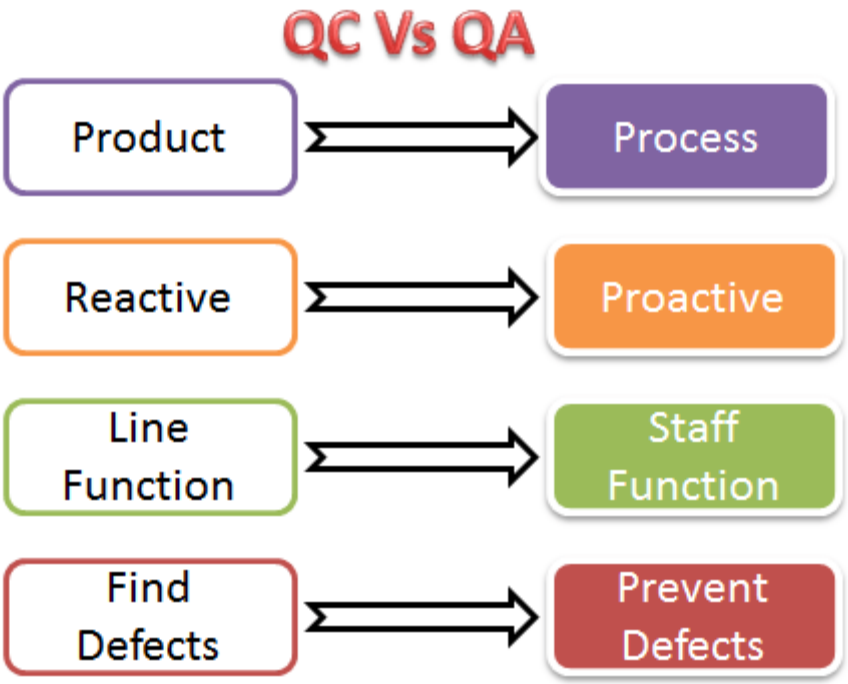
Quality control popularly abbreviated as QC. It is a Software Engineering process used to ensure quality in a product or a service. It does not deal with the processes used to create a product; rather it examines the quality of the "end products" and the final outcome.

The main aim of Quality control is to check whether the products meet the specifications and requirements of the customer. If an issue or problem is identified, it needs to be fixed before delivery to the customer.

QC also evaluates people on their quality level skill sets and imparts training and certifications. This evaluation is required for the service based organization and helps provide "perfect" service to the customers.

Difference between Quality Control and Quality Assurance?

Sometimes, QC is confused with the QA. Quality control is to examine the product or service and check for the result. Quality Assurance in Software Engineering is to examine the processes and make changes to the processes which led to the end-product.



Examples of QC and QA activities are as follows:

Quality Control Activities	Quality Assurance Activities
Walkthrough	Quality Audit
Testing	Defining Process
Inspection	Tool Identification and selection
Checkpoint review	Training of Quality Standards and Processes

The above activities are concerned with Quality Assurance and Control mechanisms for any product and not essentially software. With respect to software

- QA becomes SQA (Software Quality Assurance)
- QC becomes Software Testing.

Differences between SQA and Software Testing

Following table explains on differences between SQA and Software Testing:

SQA	Software Testing
Software Quality Assurance is about engineering process that ensures quality	Software Testing is to test a product for problems before the product goes live
Involves activities related to the implementation of processes, procedures, and standards. Example – Audits Training	Involves actives concerning verification of product Example – Review Testing
Process focused	Product focused
Preventive technique	Corrective technique
Proactive measure	Reactive measure
The scope of SQA applied to all products that will be created by the organization	The scope of Software Testing applies to a particular product being tested.

Best practices for Quality Assurance:

- Create a Robust Testing Environment
- Select release criteria carefully
- Apply automated testing to high-risk areas to save money. It helps to fasten the entire process.
- Allocate Time Appropriately for each process
- It is important to prioritize bugs fixes based on software usage
- Form dedicated security and performance testing team
- Simulate customer accounts similar to a production environment

Quality Assurance Functions:

There are 5 primary Quality Assurance Functions:

1. **Technology transfer:** This function involves getting a product design document as well as trial and error data and its evaluation. The documents are distributed, checked and approved
2. **Validation:** Here validation master plan for the entire system is prepared. Approval of test criteria for validating product and process is set. Resource planning for execution of a validation plan is done.
3. **Documentation:** This function controls the distribution and archiving of documents. Any change in a document is made by adopting the proper change control procedure. Approval of all types of documents.
4. **Assuring Quality of products**
5. **Quality improvement plans**

Quality Assurance Certifications:

There are several certifications available in the industry to ensure that Organizations follow Standards Quality Processes. Customers make this as qualifying criteria while selecting a software vendor.

ISO 9000

This standard was first established in 1987, and it is related to Quality Management Systems. This helps the organization ensure quality to their customers and other stakeholders. An organization who wishes to be certified as ISO 9000 is audited based on their functions, products, services and their processes. The main objective is to review and verify whether the organization is following the process as expected and check whether existing processes need improvement.

This certification helps –

- Increase the profit of the organization
- Improves Domestic and International trade
- Reduces waste and increase the productivity of the employees
- Provide Excellent customer satisfaction

CMMI level

The **Capability Maturity Model Integrated (CMMI)** is a process improvement approach developed specially for software process improvement. It is based on the process maturity framework and used as a general aid in business processes in the Software Industry. This model is highly regarded and widely used in Software Development Organizations.

CMMI has 5 levels. An organization is certified at CMMI level 1 to 5 based on the maturity of their Quality Assurance Mechanisms.

- Level 1 – **Initial:** In this stage the quality environment is unstable. Simply, no processes have been followed or documented
- Level 2 – **Repeatable:** Some processes are followed which are repeatable. This level ensures processes are followed at the project level.
- Level 3 – **Defined:** Set of processes are defined and documented at the organizational level. Those defined processes are subject to some degree of improvement.
- Level 4 – **Managed:** This level uses process metrics and effectively controls the processes that are followed.
- Level 5 – **Optimizing:** This level focuses on the continuous improvements of the processes through learning & innovation.

Test Maturity Model (TMM):

This model assesses the maturity of processes in a Testing Environment. Even this model has 5 levels, defined below-

- Level 1 – **Initial**: There is no quality standard followed for testing processes and only ad-hoc methods are used at this level
- Level 2 – **Definition:** Defined process. Preparation of test strategy, plans, test cases are done.
- Level 3 – **Integration:** Testing is carried out throughout the software development lifecycle (SDLC) – which is nothing but integration with the development activities, E.g., V- Model.
- Level 4 – **Management and Measurement:** Review of requirements and designs takes place at this level and criteria has been set for each level of testing
- Level 5 – **Optimization:** Many preventive techniques are used for testing processes, and tool support(Automation) is used to improve the testing standards and processes.

Conclusion:

Quality Assurance is to check whether the product developed is fit for use. For that, Organization should have processes and standards to be followed which need to be improved on a periodic basis. It concentrates mainly on the quality of product/service that we are providing to the customers during or after implementation of software.

Coding and unit Testing

Unit testing

Unit Testing is a software testing technique by means of which individual units of software i.e. group of computer program modules, usage procedures and operating procedures are tested to determine whether they are suitable for use or not. It is a testing method using which every independent modules are tested to determine if there are any issue by the developer himself. It is correlated with functional correctness of the independent modules.

Unit Testing is defined as a type of software testing where individual components of a software are tested.

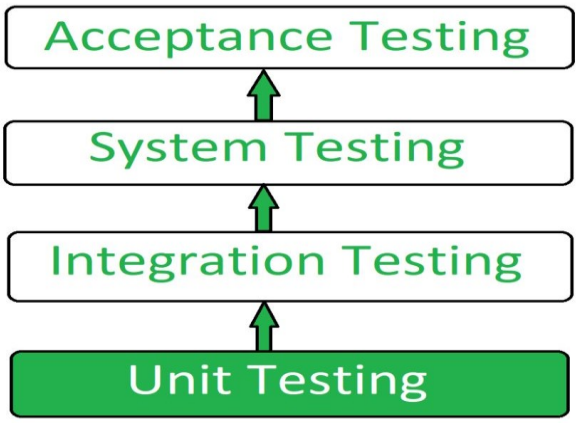
Unit Testing of software product is carried out during the development of an application. An individual component may be either an individual function or a procedure. Unit Testing is typically performed by the developer.

In SDLC or V Model, Unit testing is first level of testing done before integration testing. Unit testing is such type of testing technique that is usually performed by the developers. Although due to reluctance of developers to tests, quality assurance engineers also do unit testing.

Objective of Unit Testing:

The objective of Unit Testing is:

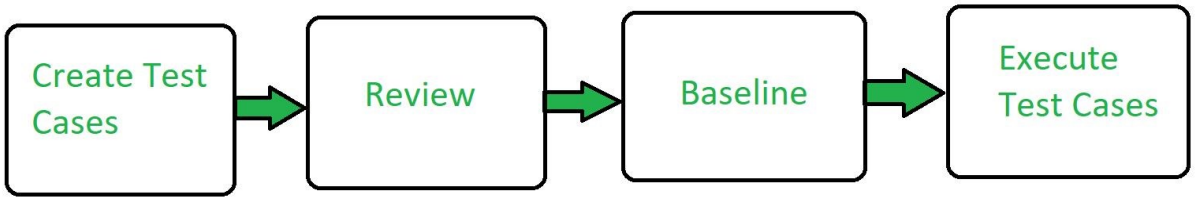
1. To isolate a section of code.
2. To verify the correctness of code.
3. To test every function and procedure.
4. To fix bug early in development cycle and to save costs.
5. To help the developers to understand the code base and enable them to make changes quickly.
6. To help for code reuse.



Types of Unit Testing:

There are 2 type of Unit Testing: **Manual**, and **Automated**.

Workflow of Unit Testing:



Unit Testing Tools:

Here are some commonly used Unit Testing tools:

1. Jtest
2. Junit
3. NUnit
4. EMMA
5. PHPUnit

Advantages of Unit Testing:

- Unit Testing allows developers to learn what functionality is provided by a unit and how to use it to gain a basic understanding of the unit API.
- Unit testing allows the programmer to refine code and make sure the module works properly.
- Unit testing enables to test parts of the project without waiting for others to be completed.

Non execution based testing:

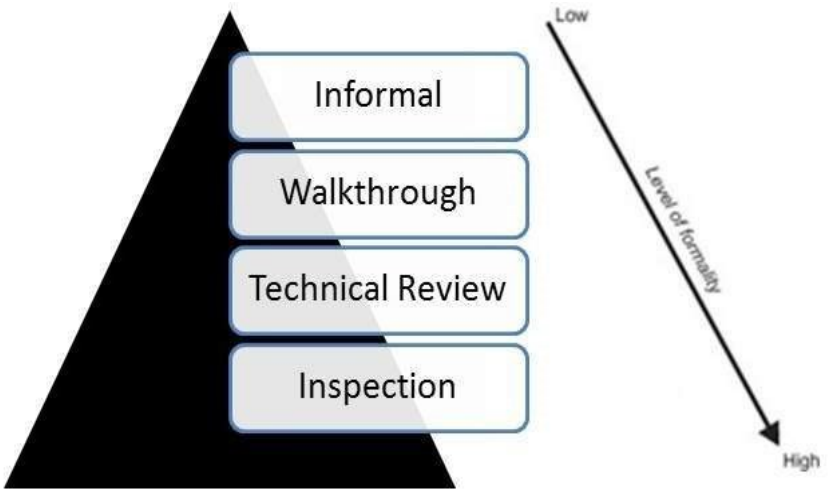
- also known as static testing
- two major practices are *reviews* and *correctness proving*
- reviews can be performed in any workflow
- goal is to detect faults as early as possible. Why?
- major risk is temptation to use information from reviews for personnel performance evaluation
- two major review practices are: *walkthroughs* and *inspections*

Code inspection

Code Inspection is the most formal type of review, which is a kind of static testing to avoid the defect multiplication at a later stage.

- The main purpose of code inspection is to find defects and it can also spot any process improvement if any.
- An inspection report lists the findings, which include metrics that can be used to aid improvements to the process as well as correcting defects in the document under review.
- Preparation before the meeting is essential, which includes reading of any source documents to ensure consistency.
- Inspections are often led by a trained moderator, who is not the author of the code.
- The inspection process is the most formal type of review based on rules and checklists and makes use of entry and exit criteria.
- It usually involves peer examination of the code and each one has a defined set of roles.
- After the meeting, a formal follow-up process is used to ensure that corrective action is completed in a timely manner.

Where Code Inspection fits in ?



Testing process

These are 11 steps software testing process is an experience based practical approach for solution to test assignment.

These are explained as following below.

Step-1: Assess Development Plan and Status –

This initiative may be prerequisite to putting together Verification, Validation, and Testing Plan wont to evaluate implemented software solution. During this step, testers challenge completeness and correctness of event plan. Based on extensiveness and completeness of Project Plan testers can estimate quantity of resources they're going to got to test implemented software solution.

Step-2: Develop the Test Plan –

Forming plan for testing will follow an equivalent pattern as any software planning process. The structure of all plans should be an equivalent, but content will vary supported degree of risk testers perceive as related to software being developed.

Step-3: Test Software Requirements –

Incomplete, inaccurate, or inconsistent requirements cause most software failures. The inability to get requirement right during requirements gathering phase can also increase cost of implementation significantly. Testers, through verification, must determine that requirements are accurate, complete, and they do not conflict with another.

Step-4: Test Software Design –

This step tests both external and internal design primarily through verification techniques. The testers are concerned that planning will achieve objectives of wants, also because design being effective and efficient on designated hardware.

Step-5: Build Phase Testing –

The method chosen to build software from internal design document will determine type and extensiveness of testers needed. As the construction becomes more automated, less testing are going to be required during this phase. However, if software is made using waterfall process, it's subject to error and will be verified. Experience has shown that it's significantly cheaper to spot defects during development phase, than through dynamic testing during test execution step.

Step-6: Execute and Record Result –
This involves testing of code during dynamic state. The approach, methods, and tools laid out in test plan are going to be wont to validate that executable code actually meets stated software requirements, and therefore the structural specifications of design.
Step-7: Acceptance Test –
Acceptance testing enables users to gauge applicability and usefulness of software in performing their day-to-day job functions. This tests what user believes software should perform, as against what documented requirements state software should perform.
Step-8: Report Test Results –
Test reporting is continuous process. It may be both oral and written. It is important that defects and concerns be reported to the appropriate parties as early as possible, so that corrections can be made at the lowest possible cost.
Step-9: The Software Installation –
Once test team has confirmed that software is prepared for production use, power to execute that software during production environment should be tested. This tests interface to operating software, related software, and operating procedures.
Step-10: Test Software Changes –
While this is often shown as Step 10, within context of performing maintenance after software is implemented, concept is additionally applicable to changes throughout implementation process. Whenever requirements changes, test plan must change, and impact of that change on software systems must be tested and evaluate.
Step-11: Evaluate Test Effectiveness –
Testing improvement can best be achieved by evaluating effectiveness of testing at top of every software test assignment. While this assessment is primarily performed by testers, it should involve developers, users of software, and quality assurance professionals if function exists within the IT organization.

Black box testing

Black box testing is a type of software testing in which the functionality of the software is not known. The testing is done without the internal knowledge of the products.
Black box testing can be done in following ways:
1. Syntax Driven Testing – This type of testing is applied to systems that can be syntactically represented by some language. For example- compilers, language that can be represented by context free grammar. In this, the test cases are generated so that each grammar rule is used at least once.
2. Equivalence partitioning – It is often seen that many type of inputs work similarly so instead of giving all of them separately we can group them together and test only one input of each group. The idea is to partition the input domain of the system into a number of equivalence classes such that each member of class works in a similar way, i.e., if a test case in one class results in some error, other members of class would also result into same error.
The technique involves two steps:

1. **Identification of equivalence class –** Partition any input domain into minimum two sets: **valid values** and **invalid values**. For example, if the valid range is 0 to 100 then select one valid input like 49 and one invalid like 104.
2. **Generating test cases –**

(i) To each valid and invalid class of input assign unique identification number.
(ii) Write test case covering all valid and invalid test case considering that no two invalid inputs mask each other.

To calculate the square root of a number, the equivalence classes will be:

(a) Valid inputs:

- Whole number which is a perfect square- output will be an integer.
- Whole number which is not a perfect square- output will be decimal number.
- Positive decimals

(b) Invalid inputs:

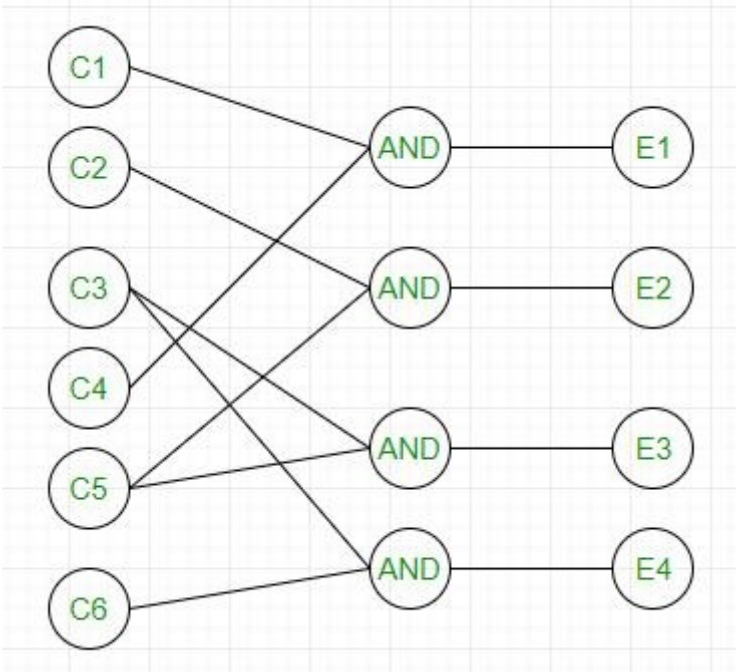
- Negative numbers(integer or decimal).
- Characters other that numbers like "a","!",";",etc.

3. Boundary value analysis – Boundaries are very good places for errors to occur. Hence if test cases are designed for boundary values of input domain then the efficiency of testing improves and probability of finding errors also increase. For example – If valid range is 10 to 100 then test for 10,100 also apart from valid and invalid inputs.

4. Cause effect Graphing – This technique establishes relationship between logical input called causes with corresponding actions called effect. The causes and effects are represented using Boolean graphs. The following steps are followed:

1. Identify inputs (causes) and outputs (effect).
2. Develop cause effect graph.
3. Transform the graph into decision table.
4. Convert decision table rules to test cases.

For example, in the following cause effect graph:



It can be converted into decision table like:

		1	2	3	4
CAUSES	C1	1	0	0	0
	C2	0	1	0	0
	C3	0	0	1	1
	C4	1	0	0	0
	C5	0	1	1	0
	C6	0	0	0	1
EFFECTS	E1	x	-	-	-
	E2	-	x	-	-
	E3	-	-	x	-
	E4	-	-	-	x

Each column corresponds to a rule which will become a test case for testing. So there will be 4 test cases.

5. Requirement based testing – It includes validating the requirements given in SRS of software system.

6. Compatibility testing – The test case result not only depend on product but also infrastructure for delivering functionality. When the infrastructure parameters are changed it is still expected to work properly. Some parameters that generally affect compatibility of software are:

1. Processor (Pentium 3,Pentium 4) and number of processors.
2. Architecture and characteristic of machine (32 bit or 64 bit).
3. Back-end components such as database servers.
4. Operating System (Windows, Linux, etc).

White box testing

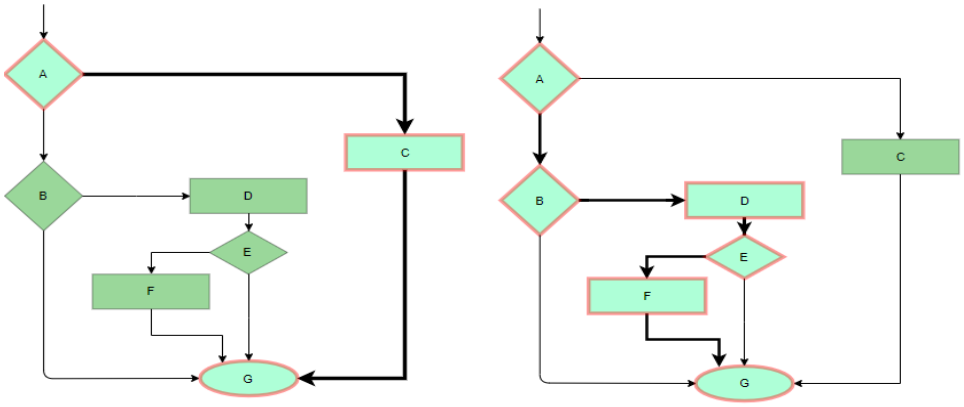
White box testing techniques analyze the internal structures the used data structures, internal design, code structure and the working of the software rather than just the functionality as in black box testing. It is also called glass box testing or clear box testing or structural testing.

Working process of white box testing:

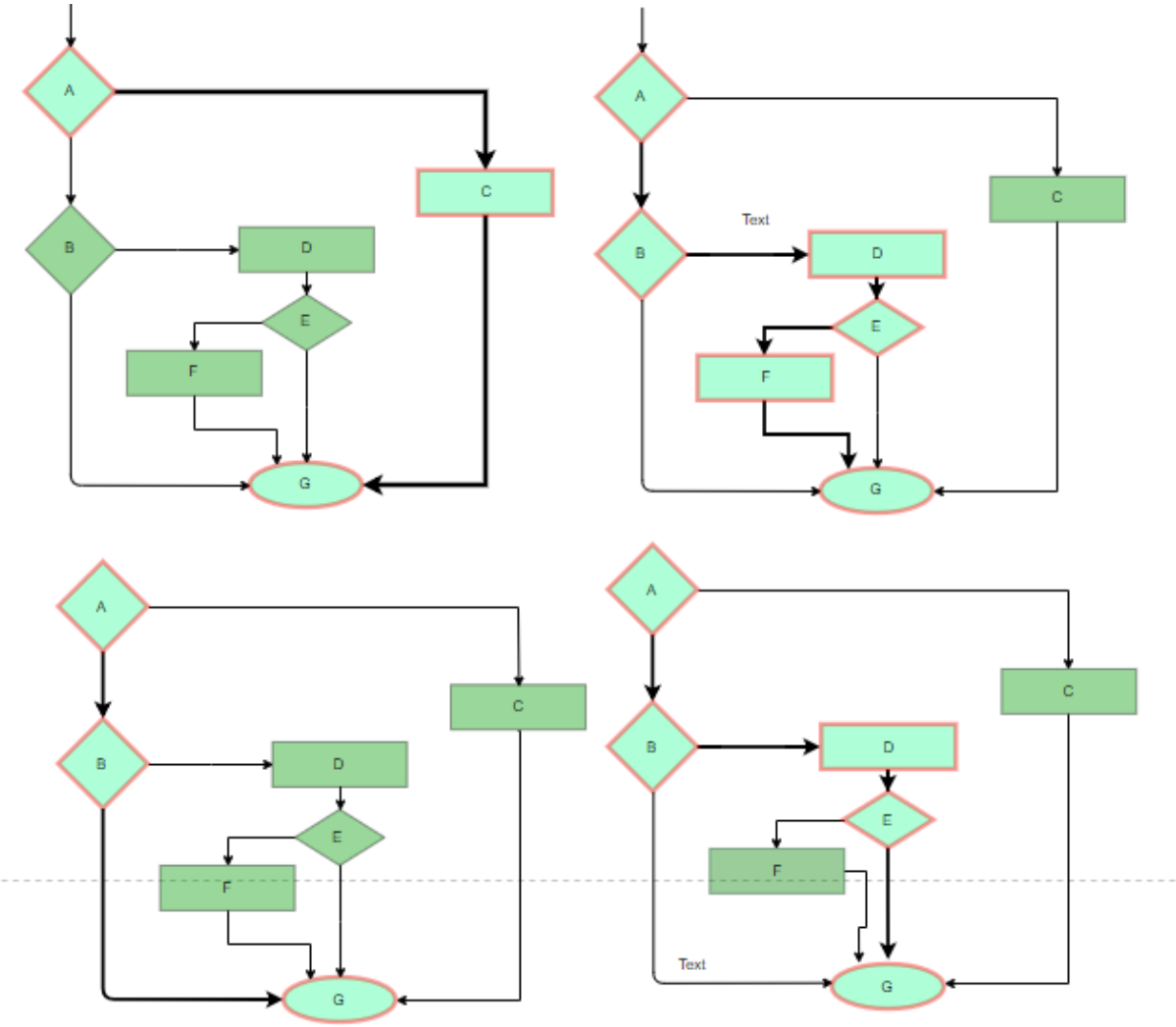
- **Input:** Requirements, Functional specifications, design documents, source code.
- **Processing:** Performing risk analysis for guiding through the entire process.
- **Proper test planning:** Designing test cases so as to cover entire code. Execute rinse-repeat until error-free software is reached. Also, the results are communicated.
- **Output:** Preparing final report of the entire testing process.

Testing techniques:

- **Statement coverage:** In this technique, the aim is to traverse all statement at least once. Hence, each line of code is tested. In case of a flowchart, every node must be traversed at least once. Since all lines of code are covered, helps in pointing out faulty code.



- **Branch Coverage:** In this technique, test cases are designed so that each branch from all decision points are traversed at least once. In a flowchart, all edges must be traversed at least once.



- **Condition Coverage:** In this technique, all individual conditions must be covered as shown in the following example:
    1. READ X, Y
    2. IF(X == 0 || Y == 0)
    3. PRINT '0'

In this example, there are 2 conditions: X == 0 and Y == 0. Now, test these conditions get TRUE and FALSE as their values. One possible example would be:

4. #TC1 – X = 0, Y = 55
5. #TC2 – X = 5, Y = 0

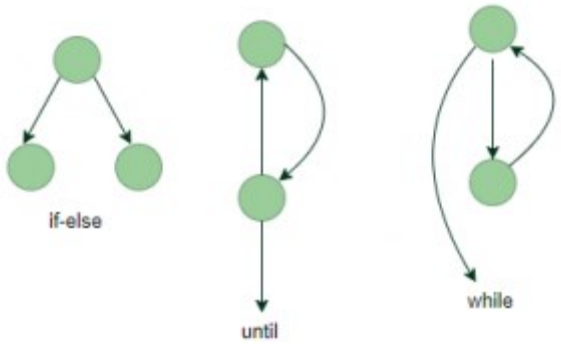
- **Multiple Condition Coverage:** In this technique, all the possible combinations of the possible outcomes of conditions are tested at least once. Let's consider the following example:

1. READ X, Y
2. IF(X == 0 || Y == 0)
3. PRINT '0'

1. #TC1: X = 0, Y = 0
2. #TC2: X = 0, Y = 5
3. #TC3: X = 55, Y = 0
4. #TC4: X = 55, Y = 5

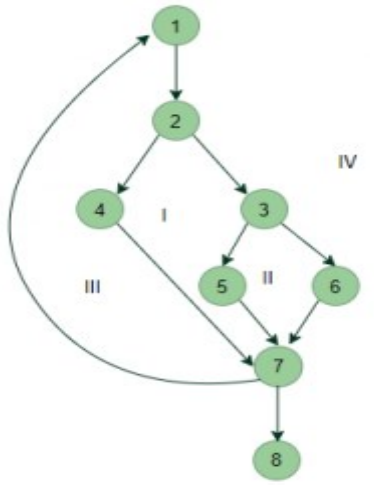
Hence, four test cases required for two individual conditions.
Similarly, if there are n conditions then 2^n test cases would be required.

- Basis Path Testing: In this technique, control flow graphs are made from code or flowchart and then Cyclomatic complexity is calculated which defines the number of independent paths so that the minimal number of test cases can be designed for each independent path.
  Steps:
  1. Make the corresponding control flow graph
  2. Calculate the cyclomatic complexity
  3. Find the independent paths
  4. Design test cases corresponding to each independent path
- Flow graph notation: It is a directed graph consisting of nodes and edges. Each node represents a sequence of statements, or a decision point. A predicate node is the one that represents a decision point that contains a condition after which the graph splits. Regions are bounded by nodes and edges.



- **Cyclomatic Complexity:** It is a measure of the logical complexity of the software and is used to define the number of independent paths. For a graph G, V(G) is its cyclomatic complexity.
  Calculating V(G):
  1. V(G) = P + 1, where P is the number of predicate nodes in the flow graph
  2. V(G) = E – N + 2, where E is the number of edges and N is the total number of nodes
  3. V(G) = Number of non-overlapping regions in the graph

Example:



V(G) = 4 (Using any of the above formulae)

No of independent paths = 4

1. #P1: 1 – 2 – 4 – 7 – 8
2. #P2: 1 – 2 – 3 – 5 – 7 – 8
3. #P3: 1 – 2 – 3 – 6 – 7 – 8
4. #P4: 1 – 2 – 4 – 7 – 1 – . . . – 7 – 8

- Loop Testing: Loops are widely used and these are fundamental to many algorithms hence, their testing is very important. Errors often occur at the beginnings and ends of loops.
  1. Simple loops: For simple loops of size n, test cases are designed that:
     - Skip the loop entirely
     - Only one pass through the loop
     - 2 passes
     - m passes, where m < n
     - n-1 ans n+1 passes
  2. Nested loops: For nested loops, all the loops are set to their minimum count and we start from the innermost loop. Simple loop tests are conducted for the innermost loop and this is worked outwards till all the loops have been tested.
  3. Concatenated loops: Independent loops, one after another. Simple loop tests are applied for each.
     If they're not independent, treat them like nesting.

Advantages:

1. White box testing is very thorough as the entire code and structures are tested.
2. It results in the optimization of code removing error and helps in removing extra lines of code.
3. It can start at an earlier stage as it doesn't require any interface as in case of black box testing.
4. Easy to automate.

Disadvantages:

1. Main disadvantage is that it is very expensive.
2. Redesign of code and rewriting code needs test cases to be written again.
3. Testers are required to have in-depth knowledge of the code and programming language as opposed to black box testing.

4. Missing functionalities cannot be detected as the code that exists is tested.
5. Very complex and at times not realistic.

Metric

Software Metrics

A software metric is a measure of software characteristics which are measurable or countable. Software metrics are valuable for many reasons, including measuring software performance, planning work items, measuring productivity, and many other uses.

Within the software development process, many metrics are that are all connected. Software metrics are similar to the four functions of management: Planning, Organization, Control, or Improvement.

Classification of Software Metrics

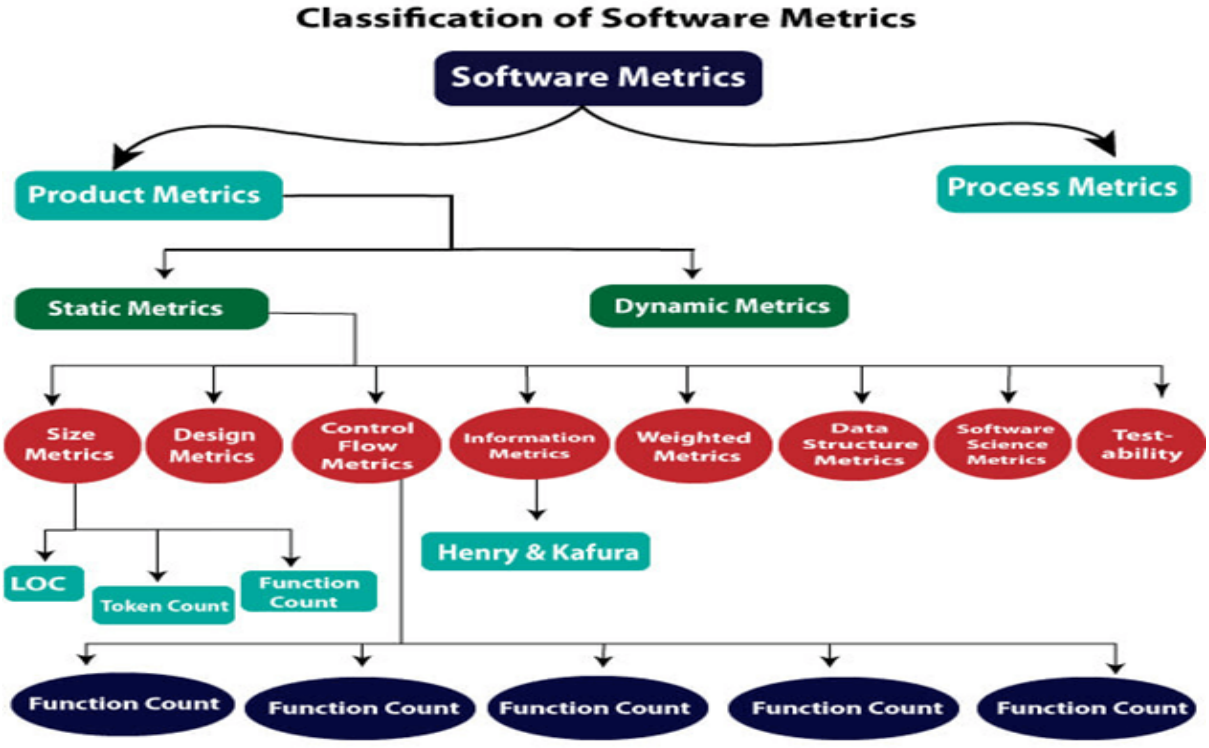
Software metrics can be classified into two types as follows:

1. Product Metrics: These are the measures of various characteristics of the software product. The two important software characteristics are:

1. Size and complexity of software.
2. Quality and reliability of software.

These metrics can be computed for different stages of SDLC.

2. Process Metrics: These are the measures of various characteristics of the software development process. For example, the efficiency of fault detection. They are used to measure the characteristics of methods, techniques, and tools that are used for developing software.



Types of Metrics

- **Internal metrics:** Internal metrics are the metrics used for measuring properties that are viewed to be of greater importance to a software developer. For example, Lines of Code (LOC) measure.

- **External metrics:** External metrics are the metrics used for measuring properties that are viewed to be of greater importance to the user, e.g., portability, reliability, functionality, usability, etc.
- **Hybrid metrics:** Hybrid metrics are the metrics that combine product, process, and resource metrics. For example, cost per FP where FP stands for Function Point Metric.
- **Project metrics:** Project metrics are the metrics used by the project manager to check the project's progress. Data from the past projects are used to collect various metrics, like time and cost; these estimates are used as a base of new software. Note that as the project proceeds, the project manager will check its progress from time-to-time and will compare the effort, cost, and time with the original effort, cost and time. Also understand that these metrics are used to decrease the development costs, time efforts and risks. The project quality can also be improved. As quality improves, the number of errors and time, as well as cost required, is also reduced.

Advantage of Software Metrics

- Comparative study of various design methodology of software systems. For analysis, comparison, and critical study of different programming language concerning their characteristics.
- In comparing and evaluating the capabilities and productivity of people involved in software development.
- In the preparation of software quality specifications.
- In the verification of compliance of software systems requirements and specifications.
- In making inference about the effort to be put in the design and development of the software systems.
- In getting an idea about the complexity of the code.
- In taking decisions regarding further division of a complex module is to be done or not.
- In guiding resource manager for their proper utilization.
- In comparison and making design trade-offs between software development and maintenance cost.
- In providing feedback to software managers about the progress and quality during various phases of the software development life cycle.
- In the allocation of testing resources for testing the code.

Disadvantage of Software Metrics

- The application of software metrics is not always easy, and in some cases, it is difficult and costly.
- The verification and justification of software metrics are based on historical/empirical data whose validity is difficult to verify.
- These are useful for managing software products but not for evaluating the performance of the technical staff.
- The definition and derivation of Software metrics are usually based on assuming which are not standardized and may depend upon tools available and working environment.

Most of the predictive models rely on estimates of certain variables which are often not known precisely.

Debugging

In the context of software engineering, debugging is the process of fixing a bug in the software. In other words, it refers to identifying, analyzing and removing errors. This activity begins after the software fails to execute properly and concludes by solving the problem and successfully testing the software. It is considered to be an extremely complex and tedious task because errors need to be resolved at all stages of debugging.

Debugging Process: Steps involved in debugging are:

- Problem identification and report preparation.
- Assigning the report to software engineer to the defect to verify that it is genuine.
- Defect Analysis using modeling, documentations, finding and testing candidate flaws, etc.
- Defect Resolution by making required changes to the system.
- Validation of corrections.

Debugging Strategies:

1. Study the system for the larger duration in order to understand the system. It helps debugger to construct different representations of systems to be debugging depends on the need. Study of the system is also done actively to find recent changes made to the software.
2. Backwards analysis of the problem which involves tracing the program backward from the location of failure message in order to identify the region of faulty code. A detailed study of the region is conducting to find the cause of defects.

1. Forward analysis of the program involves tracing the program forwards using breakpoints or print statements at different points in the program and studying the results. The region where the wrong outputs are obtained is the region that needs to be focused to find the defect.
2. Using the past experience of the software debug the software with similar problems in nature. The success of this approach depends on the expertise of the debugger.

Debugging Tools:

Debugging tool is a computer program that is used to test and debug other programs. A lot of public domain software like gdb and dbx are available for debugging. They offer console-based command line interfaces. Examples of automated debugging tools include code based tracers, profilers, interpreters, etc.

Some of the widely used debuggers are:

- Radare2
- WinDbg
- Valgrind

Difference Between Debugging and Testing:

Debugging is different from testing. Testing focuses on finding bugs, errors, etc whereas debugging starts after a bug has been identified in the software. Testing is used to ensure that the program is correct and it was supposed to do with a certain minimum success rate. Testing can be manual or automated. There are several different types of testing like unit testing, integration testing, alpha and beta testing, etc.

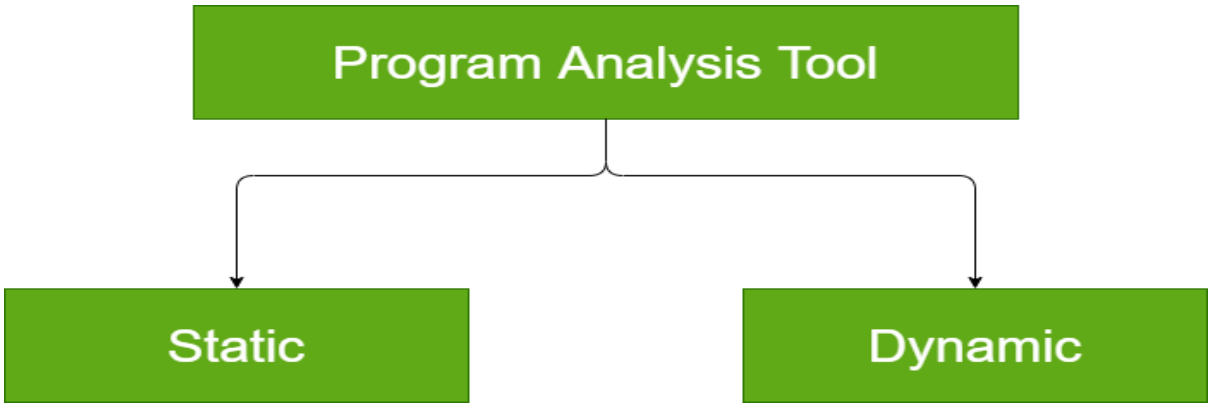
Debugging requires a lot of knowledge, skills, and expertise. It can be supported by some automated tools available but is more of a manual process as every bug is different and requires a different technique, unlike a pre-defined testing mechanism.

Program analysis tool

Program Analysis Tool is an automated tool whose input is the source code or the executable code of a program and the output is the observation of characteristics of the program. It gives various characteristics of the program such as its size, complexity, adequacy of commenting, adherence to programming standards and many other characteristics.

Classification of Program Analysis Tools:

Program Analysis Tools are classified into two categories:



1. Static Program Analysis Tools:

Static Program Analysis Tool is such a program analysis tool that evaluates and computes various characteristics of a software product without executing it. Normally, static program analysis tools analyze some structural representation of a program to reach a certain analytical conclusion. Basically some structural properties are analyzed using static program analysis tools.

The structural properties that are usually analyzed are:

1. Whether the coding standards have been fulfilled or not.
2. Some programming errors such as uninitialized variables.
3. Mismatch between actual and formal parameters.
4. Variables that are declared but never used.

Code walkthroughs and code inspections are considered as static analysis methods but static program analysis tool is used to designate automated analysis tools. Hence, a compiler can be considered as a static program analysis tool.

2. Dynamic Program Analysis Tools:

Dynamic Program Analysis Tool is such type of program analysis tool that require the program to be executed and its actual behavior to be observed. A dynamic program analyzer basically implements the code. It adds additional statements in the source code to collect the traces of program execution. When the code is executed, it allows us to observe the behavior of the software for different test cases. Once the software is tested and its behavior is observed, the dynamic program analysis tool performs a post execution analysis and produces reports which describe the structural coverage that has been achieved by the complete testing process for the program.

For example, the post execution dynamic analysis report may provide data on extent statement, branch and path coverage achieved.
The results of dynamic program analysis tools are in the form of a histogram or a pie chart. It describes the structural coverage obtained for different modules of the program. The output of a dynamic program analysis tool can be stored and printed easily and provides evidence that complete testing has been done. The result of dynamic analysis is the extent of testing performed as white box testing. If the testing result is not satisfactory then more test cases are designed and added to the test scenario. Also dynamic analysis helps in elimination of redundant test cases.

Integration Testing

Integration testing is the process of testing the interface between two software units or module. It's focus on determining the correctness of the interface. The purpose of the integration testing is to expose faults in the interaction between integrated units. Once all the modules have been unit tested, integration testing is performed.

Integration test approaches –
There are four types of integration testing approaches. Those approaches are the following:
1. Big-Bang Integration Testing –
It is the simplest integration testing approach, where all the modules are combining and verifying the functionality after the completion of individual module testing. In simple words, all the modules of the system are simply put together and tested. This approach is practicable only for very small systems. If once an error is found during the integration testing, it is very difficult to localize the error as the error may potentially belong to any of the modules being integrated. So, debugging errors reported during big bang integration testing are very expensive to fix.
Advantages:
- It is convenient for small systems.

Disadvantages:
- There will be quite a lot of delay because you would have to wait for all the modules to be integrated.
- High risk critical modules are not isolated and tested on priority since all modules are tested at once.

2. Bottom-Up Integration Testing –
In bottom-up testing, each module at lower levels is tested with higher modules until all modules are tested. The primary purpose of this integration testing is, each subsystem is to test the interfaces among various modules making up the subsystem. This integration testing uses test drivers to drive and pass appropriate data to the lower level modules.
Advantages:
- In bottom-up testing, no stubs are required.
- A principle advantage of this integration testing is that several disjoint subsystems can be tested simultaneously.

Disadvantages:
- Driver modules must be produced.
- In this testing, the complexity that occurs when the system is made up of a large number of small subsystem.

3. Top-Down Integration Testing –

Top-down integration testing technique used in order to simulate the behaviour of the lower-level modules that are not yet integrated.In this integration testing, testing takes place from top to bottom. First high-level modules are tested and then low-level modules and finally integrating the low-level modules to a high level to ensure the system is working as intended.

Advantages:

- Separately debugged module.
- Few or no drivers needed.
- It is more stable and accurate at the aggregate level.

Disadvantages:

- Needs many Stubs.
- Modules at lower level are tested inadequately.

4. Mixed Integration Testing –

A mixed integration testing is also called sandwiched integration testing. A mixed integration testing follows a combination of top down and bottom-up testing approaches. In top-down approach, testing can start only after the top-level module have been coded and unit tested. In bottom-up approach, testing can start only after the bottom level modules are ready. This sandwich or mixed approach overcomes this shortcoming of the top-down and bottom-up approaches. A mixed integration testing is also called sandwiched integration testing.

Advantages:

- Mixed approach is useful for very large projects having several sub projects.
- This Sandwich approach overcomes this shortcoming of the top-down and bottom-up approaches.

Disadvantages:

- For mixed integration testing, require very high cost because one part has Top-down approach while another part has bottom-up approach.
- This integration testing cannot be used for smaller system with huge interdependence between different modules.

System testing

System Testing is a type of software testing that is performed on a complete integrated system to evaluate the compliance of the system with the corresponding requirements.

In system testing, integration testing passed components are taken as input. The goal of integration testing is to detect any irregularity between the units that are integrated together. System testing detects defects within both the integrated units and the whole system. The result of system testing is the observed behaviour of a component or a system when it is tested.

System Testing is carried out on the whole system in the context of either system requirement specifications or functional requirement specifications or in the context of both. System testing tests the design and behaviour of the system and also the expectations of the customer. It is performed to test the system beyond the bounds mentioned in the software requirements specification (SRS).

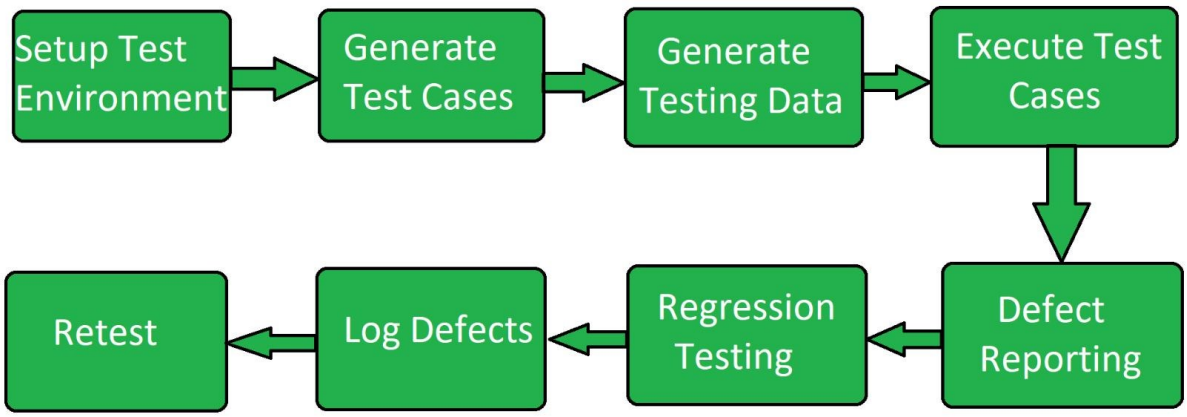
System Testing is basically performed by a testing team that is independent of the development team that helps to test the quality of the system impartial. It has both functional and non-functional testing.

System Testing is a black-box testing.

System Testing is performed after the integration testing and before the acceptance testing.

System Testing Process:
System Testing is performed in the following steps:

- **Test Environment Setup:**
  Create testing environment for the better quality testing.
- **Create Test Case:**
  Generate test case for the testing process.
- **Create Test Data:**
  Generate the data that is to be tested.
- **Execute Test Case:**
  After the generation of the test case and the test data, test cases are executed.
- **Defect Reporting:**
  Defects in the system are detected.
- **Regression Testing:**
  It is carried out to test the side effects of the testing process.
- **Log Defects:**
  Defects are fixed in this step.
- **Retest:**
  If the test is not successful then again test is performed.



Types of System Testing:

- **Performance Testing:**
  Performance Testing is a type of software testing that is carried out to test the speed, scalability, stability and reliability of the software product or application.
- **Load Testing:**
  Load Testing is a type of software Testing which is carried out to determine the behaviour of a system or software product under extreme load.
- **Stress Testing:**
  Stress Testing is a type of software testing performed to check the robustness of the system under the varying loads.
- **Scalability Testing:**
  Scalability Testing is a type of software testing which is carried out to check the performance of a software application or system in terms of its capability to scale up or scale down the number of user request load.

Testing distributed implementation

The way you test your distributed system really depends on the nature of your system.
Distributed systems serve a specific need, so its architecture is built in a way to serve that business need. Chances are, the testing methods vary as well, and I haven't even mentioned the number of independent services that also influences the testing methodology.

How do you test your distributed system?
We follow a pyramid approach.
We have unit tests at the bottom. It doesn't matter if you're distributed or not; units are always tested at full coverage
Then comes the integration layer for the distributed components. We follow a very service-oriented interface; the big components we built (whether its lambda or a trigger on the database) make integration tests a lot easier here.
System-level testing: When the distributed nature comes in, we typically follow well-set out standards.

Tools
We use different tools for different environments. Most of our lambda is written in Java. Integration tests are built using Java framework, while our server layer is mostly running ruby (ruby's stack). SageMaker is mostly written in Python but that side is a little bit different, but tests run in Python.
We use whatever language is appropriate.
We used to write our own distributed code that process the data, but that introduced a lot of headaches, because you have to deal with scaling, failovers and fatal tolerance. A year ago, we moved to lambda.
Since we switched to lambda, it allowed application developers to focus on business logic while alleviating the low-level concerns to the platform providers. It makes testing so much easier since you don't have to test multi threadings and fillovers.
testing is focused on 3 different phases.
Phase 1: Pure engineering tests, including code quality, static code analysis. We do a bunch of engineering tests. These are not customer-focused scenarios; these are basically unit tests. It doesn't care about business logic but rather coding practices. It happens on an ongoing basis, and nothing gets merged until this basic test is done.
Phase 2: Business tests or integration tests. We have a bunch of use case scenarios. We used to do it as part of the unit tests and we realized it's impossible to do that with a distributed system because everything is changing at the same time, so it's hard to test across the entire system.
If you want to run the test through the entire system, it would run for 8-12 hours. So, we segmented our scenarios into core and basic ones.
The first thing to do is to test core scenarios, checking for fundamental issues. Then we run basic tests, which cover most of the scenarios but not all of them (for example: not all operating systems), then we run tests on the whole system, which takes 8-12 hours.
Phase 3: Only before launch, we run a bunch of stress tests.

Testing of real time system

Real time system means that the system is subjected to real time, i.e., response should be guaranteed within a specified timing constraint or system should meet the specified deadline. For example: flight control system, real time monitors etc.

Types of real time systems based on timing constraints:

1. **Hard real time system –**

This type of system can never miss its deadline. Missing the deadline may have disastrous consequences. The usefulness of result produced by a hard real time system decreases abruptly and may become negative if tardiness increases. Tardiness means how late a real time system completes its task with respect to its deadline. Example: Flight controller system.

2. **Soft real time system –**

This type of system can miss its deadline occasionally with some acceptably low probability. Missing the deadline have no disastrous consequences. The usefulness of result produced by a soft real time system decreases gradually with increase in tardiness. Example: Telephone switches.

Reference model of real time system: Our reference model is characterized by three elements:

1. **A workload model:** It specifies the application supported by system.
2. **A resource model:** It specifies the resources available to the application.
3. **Algorithms:** It specifies how the application system will use resources.

Terms related to real time system:

- **Job –** A job is a small piece of work that can be assigned to a processor and may or may not require resources.
- **Task –** A set of related jobs that jointly provide some system functionality.
- **Release time of a job –** It is the time at which job becomes ready for execution.
- **Execution time of a job –** It is the time taken by job to finish its execution.
- **Deadline of a job –** It is the time by which a job should finish its execution. Deadline is of two types: absolute deadline and relative deadline.
- **Response time of a job –** It is the length of time from release time of a job to the instant when it finishes.
- Maximum allowable response time of a job is called its relative deadline.
- Absolute deadline of a job is equal to its relative deadline plus its release time.
- Processors are also known as active resources. They are essential for execution of a job. A job must have one or more processors in order to execute and proceed towards completion. Example: computer, transmission links.
- Resources are also known as passive resources. A job may or may not require a resource during its execution. Example: memory, mutex
- Two resources are identical if they can be used interchangeably else they are heterogeneous.

Acceptance testing some general issue associated with testing

Acceptance Testing is a method of software testing where a system is tested for acceptability. The major aim of this test is to evaluate the compliance of the system with the business requirements and assess whether it is acceptable for delivery or not.

Standard Definition of Acceptance Testing:

Acceptance Testing is the last phase of software testing performed after System Testing and before making the system available for actual use.

Types of Acceptance Testing:

1. **User Acceptance Testing (UAT):**
   User acceptance testing is used to determine whether the product is working for the user correctly. Specific requirements which are quite often used by the customers are primarily picked for the testing purpose. This is also termed as *End-User* Testing.
2. **Business Acceptance Testing (BAT):**
   BAT is used to determine whether the product meets the business goals and purposes or not. BAT mainly focuses on business profits which are quite challenging due to the changing market conditions and new technologies so that the current implementation may have to being changed which result in extra budgets.
3. **Contract Acceptance Testing (CAT):**
   CAT is a contract which specifies that once the product goes live, within a predetermined period, the acceptance test must be performed and it should pass all the acceptance use cases.
   Here is a contract termed as Service Level Agreement (SLA), which includes the terms where the payment will be made only if the Product services are in-line with all the requirements, which means the contract is fulfilled.

Sometimes, this contract happens before the product goes live. There should be a well defined contract in terms of the period of testing, areas of testing, conditions on issues encountered at later stages, payments, etc.

1. **Regulations Acceptance Testing (RAT):**
   RAT is used to determine whether the product violates the rules and regulations that are defined by the government of the country where it is being released. This may be unintentional but will impact negatively on the business.
   Generally, the product or application that is to be released in the market, has to go under RAT, as different countries or regions have different rules and regulations defined by its governing bodies. If any rules and regulations are violated for any country then that country or the specific region then the product will not be released in that country or region. If the product is released even though there is a violation then only the vendors of the product will be directly responsible.
2. **Operational Acceptance Testing (OAT):**
   OAT is used to determine the operational readiness of the product and is a non-functional testing. It mainly includes testing of recovery, compatibility, maintainability, reliability etc. OAT assures the stability of the product before it is released to the production.
   1. **Alpha Testing:**
      Alpha testing is used to determine the product in the development testing environment by a specialized testers team usually called alpha testers.
   2. **Beta Testing:**
      Beta testing is used to assess the product by exposing it to the real end-users, usually called beta testers in their environment. Feedback is collected from the users and the defects are fixed. Also, this helps in enhancing the product to give a rich user experience.

Use of Acceptance Testing:

- To find the defects missed during the functional testing phase.
- How well the product is developed.
- A product is what actually the customers need.

- Feedbacks help in improving the product performance and user experience.
- Minimize or eliminate the issues arising from the production.

Recovery testing

Recovery testing is a type of system testing which aims at testing whether a system can recover from failures or not. The technique involves failing the system and then verifying that the system recovery is performed properly.

To ensure that a system is fault-tolerant and can recover well from failures, recovery testing is important to perform. A system is expected to recover from faults and resume its work within a pre-specified time period. Recovery testing is essential for any mission-critical system, for example, the defense systems, medical devices, etc. In such systems, there is a strict protocol that is imposed on how and within what time period the system should recover from failure and how the system should behave during the failure.

A system or software should be recovery tested for failures like :

- Power supply failure
- The external server is unreachable
- Wireless network signal loss
- Physical conditions
- The external device not responding
- The external device is not responding as expected, etc.

Steps to be performed before executing a Recovery Test :

A tester must ensure that the following steps are performed before carrying out the Recovery testing procedure :

1. **Recovery Analysis –**
   It is important to analyze the system's ability to allocate extra resources like servers or additional CPUs. This would help to better understand the recovery-related changes that can impact the working of the system. Also, each of the possible failures, their possible impact, their severity, and how to perform them should be studied.
2. **Test Plan preparation –**
   Designing the test cases keeping in mind the environment and results obtained in recovery analysis.
3. **Test environment preparation –**
   Designing the test environment according to the recovery analysis results.
4. **Maintaining Back-up –**
   Information related to the software, like various states of the software and database should be backed up. Also, if the data is important, then the backing up of the data at multiple locations is important.
5. **Recovery personnel Allocation –**
   For the recovery testing process, it is important to allocate recovery personnel who is aware and educated enough for the recovery testing being conducted.
6. **Documentation –**
   This step emphasizes on documenting all the steps performed before and during the recovery testing so that the system can be analyzed for its performance in case of a failure.

Example of Recovery Testing :

- When a system is receiving some data over a network for processing purposes, we can stimulate software failure by unplugging the system power. After a while, we can plug in the system again and test its ability to recover and continue receiving the data from where it stopped.
- Another example could be when a browser is working on multiple sessions, we can stimulate software failure by restarting the system. After restarting the system, we can check if it recovers from the failure and reloads all the sessions it was previously working on.
- While downloading a movie over a Wifi network, if we move to a place where there is no network, then the downloading process will be interrupted. Now to check if the process recovers from the interruption and continues working as before, we move back to a place where there is a Wifi network. If the downloading resumes, then the software has a good recovery rate.

Advantages of Recovery Testing :

- **Improves the quality of the system** by eliminating the potential flaws in the system so that the system works as expected.
- Recovery testing is also referred to as **Disaster Recovery Testing**. A lot of companies have disaster recovery centers to make sure that if any of the systems is damaged or fails due to some reason, then there is back up to recover from the failure.
- **Risk elimination** is possible as the potential flaws are detected and removed from the system.
- **Improved performance** as faults are removed and the system becomes more reliable and performs better in case a failure occurs.

Disadvantages of Recovery testing :

- Recovery testing is **a time-consuming process** as it involves multiple steps and preparations before and during the process.
- The **recovery personnel must be trained** as the process of recovery testing takes place under his supervision. So, the tester needs to be trained to ensure that recovery testing is performed in the proper way. For performing recovery testing, he should have enough data and back up files to perform recovery testing.
- The **potential flaws or issues are unpredictable in a few cases**. It is difficult to point out the exact reason for the same, however, since the quality of the software must be maintained, so random test cases are created and executed to ensure such potential flaws are removed.

Security testing

Security Testing is a type of Software Testing that uncovers vulnerabilities of the system and determines that the data and resources of the system are protected from possible intruders. It ensures that the software system and application are free from any threats or risks that can cause a loss. Security testing of any system is focuses on finding all possible loopholes and weaknesses of the system which might result into the loss of information or repute of the organization.

Goal of Security Testing:

The goal of security testing is to:

- To identify the threats in the system.
- To measure the potential vulnerabilities of the system.

- To help in detecting every possible security risks in the system.
- To help developers in fixing the security problems through coding.

Principle of Security Testing:

Below are the six basic principles of security testing:

- Confidentiality
- Integrity
- Authentication
- Authorization
- Availability
- Non-repudiation

Major Focus Areas in Security Testing:

- Network Security
- System Software Security
- Client-side Application Security
- Server-side Application Security

Types of Security Testing:

1. **Vulnerability Scanning:**
   Vulnerability scanning is performed with the help of automated software to scan a system to detect the known vulnerability patterns.
2. **Security Scanning:**
   Security scanning is the identification of network and system weaknesses. Later on it provides solutions for reducing these defects or risks. Security scanning can be carried out in both manual and automated way.
3. **Penetration Testing:**
   Penetration testing is the simulation of the attack from a malicious hacker. It includes analysis of a particular system to examine for potential vulnerabilities from a malicious hacker that attempts to hack the system.
4. **Risk Assessment:**
   In risk assessment testing security risks observed in the organization are analysed. Risks are classified into three categories i.e. low, medium and high. This testing endorses controls and measures to minimize the risk.
5. **Security Auditing:**
   Security auditing is an internal inspection of applications and operating systems for security defects. An audit can also be carried out via line by line checking of code.
6. **Ethical Hacking:**
   Ethical hacking is different from malicious hacking. The purpose of ethical hacking is to expose security flaws in the organization system.
7. **Posture Assessment:**
   It combines security scanning, ethical hacking and risk assessments to provide an overall security posture of an organization.

Stress testing

Stress Testing is a software testing technique that determines the robustness of software by testing beyond the limits of normal operation. Stress testing is particularly important for critical software but is used for all types of software. Stress testing emphasizes on robustness,

availability and error handling under a heavy load rather than on what is correct behaviour under normal situations.

Stress testing is defined as a type of software testing that verifies the stability and reliability of the system. This test particularly determines the system on its robustness and error handling under extremely heavy load conditions. It even tests beyond the normal operating point and analyses how the system works under the extreme conditions. Stress testing is performed to ensure that the system would not crash under crunch situations. Stress testing is also known as ***Endurance Testing*** or ***Torture Testing***.

Characteristics of Stress Testing:

1. Stress testing analyzes the behavior of the system after a failure.
2. Stress testing makes sure that the system recovers after failure.
3. It checks whether the system works under the abnormal conditions.
4. It ensures to display appropriate error message when the system is under stress.
5. It verifies that unexpected failures do not cause security issues.
6. It verifies whether the system has saved the data before crashing or not.

Stress Testing Process:

Types of Stress Testing:

1. ***Server-client Stress Testing*:**
   In this stress testing, testing is carried out across all clients from the server.
2. ***Product Stress Testing*:**
   Product stress testing concentrates on discovering defects related to data locking and blocking, network issues and performance congestion in a software product.
3. ***Transaction Stress Testing*:**
   Transaction stress testing is performed on one or more transactions between two or more applications. It is carried out for fine-tuning and optimizing the system.
4. ***Systematic Stress Testing*:**
   Systematic stress testing is integrated testing which is used to perform test across multiple systems running on the same server. It is used to discover defects where one application data blocks another application.
5. ***Analytical Stress Testing*:**
   Analytical stress testing is performed to test the system with abnormal parameters or conditions that are unlikely to happen in a real scenario. It is carried out to find defects in unusual scenarios like a large number of users logged at the same time or database went offline when it is accessed from a website.

Stress Testing Tools:

1. Jmeter
2. Load Runner
3. Stress Tester
4. Neo load

Advantages of Stress Testing:

1.Stress testing determines the behaviour of the system after failure and ensures that system recovers quickly.

2.Stress testing ensures that system failure doesn't cause security issues.

3. Stress testing makes system work in normal as well as abnormal conditions in appropriate way.

Performance Testing

Performance Testing is a type of software testing that ensures software applications to perform properly under their expected workload. It is a testing technique carried out to determine system performance in terms of sensitivity, reactivity and stability under a particular workload.

Performance Testing is the process of analysing the quality and capability of a product. It is a testing method performed to determine the system performance in terms of speed, reliability and stability under varying workload. Performance testing is also known as ***Perf Testing***.

Performance Testing Attributes:

- **Speed:**
  It determines whether the software product responds rapidly.
- **Scalability:**
  It determines amount of load the software product can handle at a time.
- **Stability:**
  It determines whether the software product is stable in case of varying workloads.
- **Reliability:**
  It determines whether the software product is secure or not.

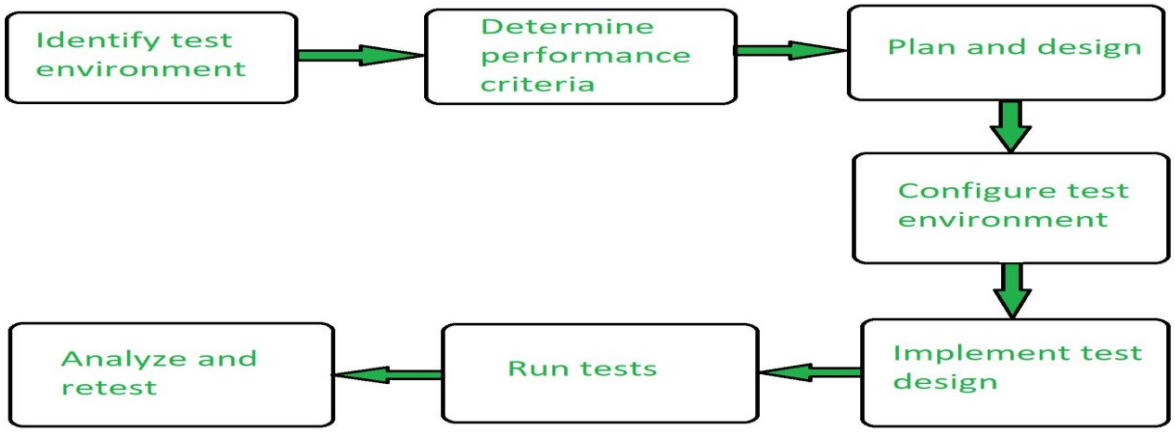
Objective of Performance Testing:

1. The objective of performance testing is to eliminate performance congestion.
2. It uncovers what is needed to be improved before the product is launched in market.
3. The objective of performance testing is to make software rapid.
4. The objective of performance testing is to make software stable and reliable.

Types of Performance Testing:

1. ***Load testing*:**
   It checks the product's ability to perform under anticipated user loads. The objective is to identify performance congestion before the software product is launched in market.
2. ***Stress testing*:**
   It involves testing a product under extreme workloads to see whether it handles high traffic or not. The objective is to identify the breaking point of a software product.
3. ***Endurance testing*:**
   It is performed to ensure the software can handle the expected load over a long period of time.
4. ***Spike testing*:**
   It tests the product's reaction to sudden large spikes in the load generated by users.
5. ***Volume testing*:**
   In volume testing large number of data is saved in a database and the overall software system's behaviour is observed. The objective is to check product's performance under varying database volumes.
6. ***Scalability testing*:**
   In scalability testing, software application's effectiveness is determined in scaling up to support an increase in user load. It helps in planning capacity addition to your software system.

Performance Testing Process:



Performance Testing Tools:

1. Jmeter
2. Open STA
3. Load Runner
4. Web Load

Software Reuse & Emerging Trends

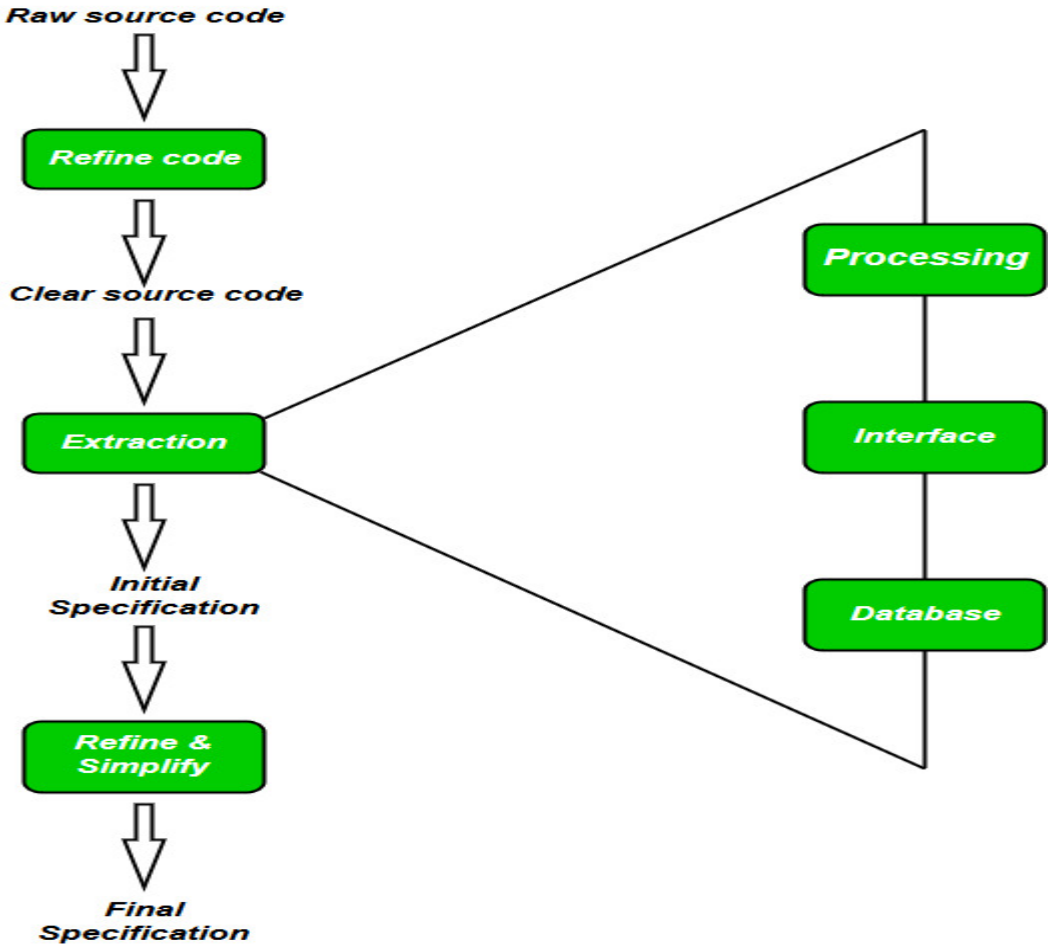
Software reverse engineering

Software Reverse Engineering is a process of recovering the design, requirement specifications and functions of a product from an analysis of its code. It builds a program database and generates information from this.

The purpose of reverse engineering is to facilitate the maintenance work by improving the understandability of a system and to produce the necessary documents for a legacy system.

Reverse Engineering Goals:

- Cope with Complexity.
- Recover lost information.
- Detect side effects.
- Synthesise higher abstraction.
- Facilitate Reuse.



Steps of Software Reverse Engineering:

1. **Collection Information:**
   This step focuses on collecting all possible information (i.e., source design documents etc.) about the software.

2. **Examining the information:**
   The information collected in step-1 as studied so as to get familiar with the system.
3. **Extracting the structure:**
   This step concerns with identification of program structure in the form of structure chart where each node corresponds to some routine.
4. **Recording the functionality:**
   During this step processing details of each module of the structure, charts are recorded using structured language like decision table, etc.
5. **Recording data flow:**
   From the information extracted in step-3 and step-4, set of data flow diagrams are derived to show the flow of data among the processes.
6. **Recording control flow:**
   High level control structure of the software is recorded.
7. **Review extracted design:**
   Design document extracted is reviewed several times to ensure consistency and correctness. It also ensures that the design represents the program.
8. **Generate documentation:**
   Finally, in this step, the complete documentation including SRS, design document, history, overview, etc. are recorded for future use.

Reverse Engineering Tools:
Reverse engineering if done manually would consume lot of time and human labour and hence must be supported by automated tools. Some of tools are given below:

- **CIAO and CIA:** A graphical navigator for software and web repositories along with a collection of Reverse Engineering tools.
- **Rigi:** A visual software understanding tool.
- **Bunch:** A software clustering/modularization tool.
- **GEN++:** An application generator to support development of analysis tools for the C++ language.
- **PBS:** Software Bookshelf tools for extracting and visualizing the architecture of programs.

Software reuse concepts

What is software reuse?
Software reuse is a term used for developing the software by using the existing software components. Some of the components that can be reuse are as follows:

- Source code
- Design and interfaces
- User manuals
- Software Documentation
- Software requirement specifications and many more.

What are the advantages of software reuse?

- **Less effort:** Software reuse requires less effort because many components use in the system are ready made components.
- **Time-saving:** Re-using the ready made components is time saving for the software team.

- **Reduce cost:** Less effort, and time saving leads to the overall cost reduction.
- Increase software productivity: when you are provided with ready made components, then you can focus on the new components that are not available just like ready made components.
- **Utilize fewer resources:** Software reuse save many sources just like effort, time, money etc.
- **Leads to a better** quality software: Software reuse save our time and we can consume our more time on maintaining software quality and assurance.

What are Commercial-off-the-shelf and Commercial-off-the-shelf components?
Commercial-off-the-shelf is ready-made software. Commercial-off-the-shelf software components are ready-made components that can be reused for a new software.

What is reuse software engineering?
Reuse software engineering is based on guidelines and principles for reusing the existing software.

What are stages of reuse-oriented software engineering?

Requirement specification:
First of all, specify the requirements. This will help to decide that we have some existing software components for the development of software or not.

Component analysis
Helps to decide that which component can be reused where.

Requirement updations / modifications.
If the requirements are changed by the customer, then still existing components are helpful for reuse or not.

Reuse System design
If the requirements are changed by the customer, then still existing system designs are helpful for reuse or not.

Development
Existing components are matching with new software or not.

Integration
Can we integrate the new systems with existing components?

System validation
To validate the system that it can be accepted by the customer or not.

software reuse success factors

1. Capturing Domain Variations
2. Easing Integration
3. Understanding Design Context
4. Effective Teamwork
5. Managing Domain Complexity

How does inheritance promote software re-usability?
Inheritance helps in the software re-usability by using the existing components of the software to create new component.
In object oriented programming protected data members are accessible in the child and so we can say that yes inheritance promote software re-usability.

Reuse of software helps reduce the need for testing?
This statement is true just for those components that are ready made and are ready to be reuse. New components must be test.

Client Server Model

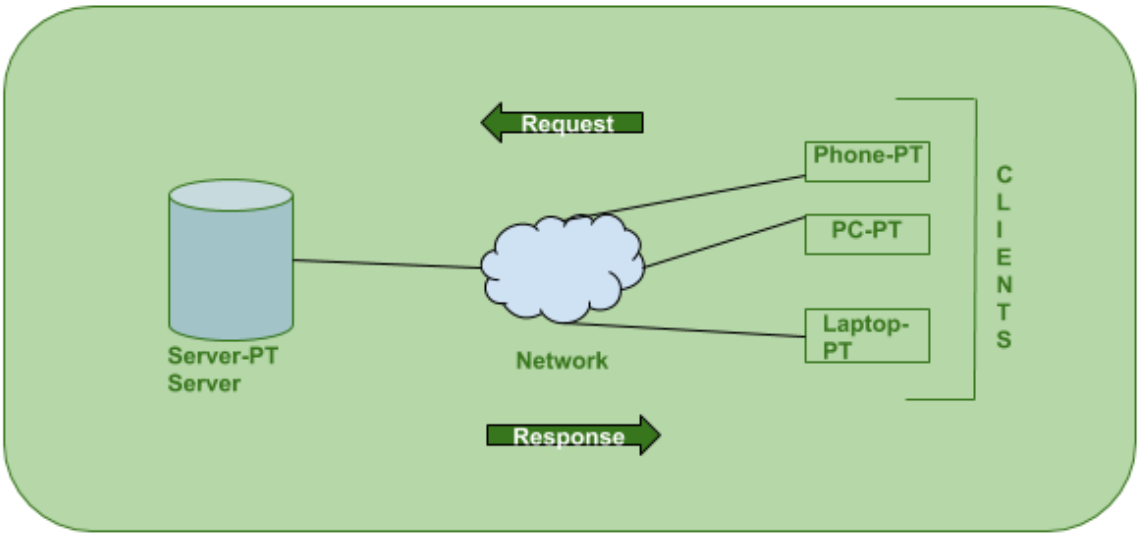
The Client-server model is a distributed application structure that partitions task or workload between the providers of a resource or service, called servers, and service requesters called clients. In the client-server architecture, when the client computer sends a request for data to the server through the internet, the server accepts the requested process and deliver the data packets requested back to the client. Clients do not share any of their resources. Examples of Client-Server Model are Email, World Wide Web, etc.

How the Client-Server Model works ?

In this article we are going to take a dive into the **Client-Server** model and have a look at how the **Internet** works via, web browsers. This article will help us in having a solid foundation of the WEB and help in working with WEB technologies with ease.

- **Client:** When we talk the word **Client**, it mean to talk of a person or an organization using a particular service. Similarly in the digital world a **Client** is a computer (**Host**) i.e. capable of receiving information or using a particular service from the service providers (**Servers**).
- **Servers:** Similarly, when we talk the word **Servers**, It mean a person or medium that serves something. Similarly in this digital world a **Server** is a remote computer which provides information (data) or access to particular services.

So, its basically the **Client** requesting something and the **Server** serving it as long as its present in the database.

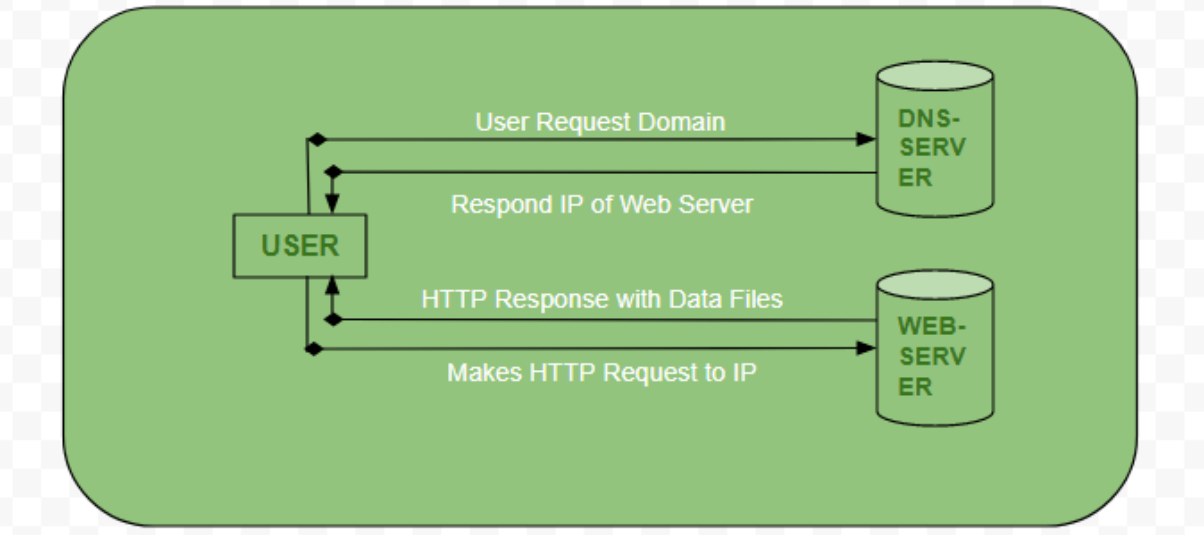


How the browser interacts with the servers ?

There are few steps to follow to interacts with the servers a client.

- User enters the **URL**(Uniform Resource Locator) of the website or file. The Browser then requests the **DNS**(DOMAIN NAME SYSTEM) Server.
- **DNS Server** lookup for the address of the **WEB Server**.
- **DNS Server** responds with the **IP address** of the **WEB Server**.
- Browser sends over an **HTTP/HTTPS** request to **WEB Server's IP** (provided by **DNS server**).

- Server sends over the necessary files of the website.
- Browser then renders the files and the website is displayed. This rendering is done with the help of **DOM** (Document Object Model) interpreter, **CSS** interpreter and **JS Engine** collectively known as the **JIT** or (Just in Time) Compilers.



Advantages of Client-Server model:

- Centralized system with all data in a single place.
- Cost efficient requires less maintenance cost and Data recovery is possible.
- The capacity of the Client and Servers can be changed separately.

Disadvantages of Client-Server model:

- Clients are prone to viruses, Trojans and worms if present in the Server or uploaded into the Server.
- Server are prone to Denial of Service (DOS) attacks.
- Data packets may be spoofed or modified during transmission.
- Phishing or capturing login credentials or other useful information of the user are common and MITM(Man in the Middle) attacks are common.

Service Oriented Architecture (SOA)

Service-Oriented Architecture (SOA) is a stage in the evolution of application development and/or integration. It defines a way to make software components reusable using the interfaces.

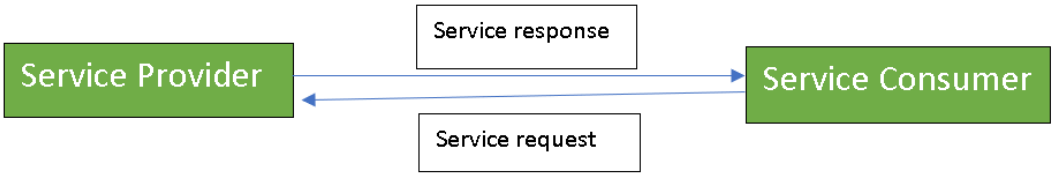
Formally, SOA is an architectural approach in which applications make use of services available in the network. In this architecture, services are provided to form applications, through a network call over the internet. It uses common communication standards to speed up and streamline the service integrations in applications. Each service in SOA is a complete business function in itself. The services are published in such a way that it makes it easy for the developers to assemble their apps using those services. Note that SOA is different from microservice architecture.

- SOA allows users to combine a large number of facilities from existing services to form applications.
- SOA encompasses a set of design principles that structure system development and provide means for integrating components into a coherent and decentralized system.

- SOA-based computing packages functionalities into a set of interoperable services, which can be integrated into different software systems belonging to separate business domains.

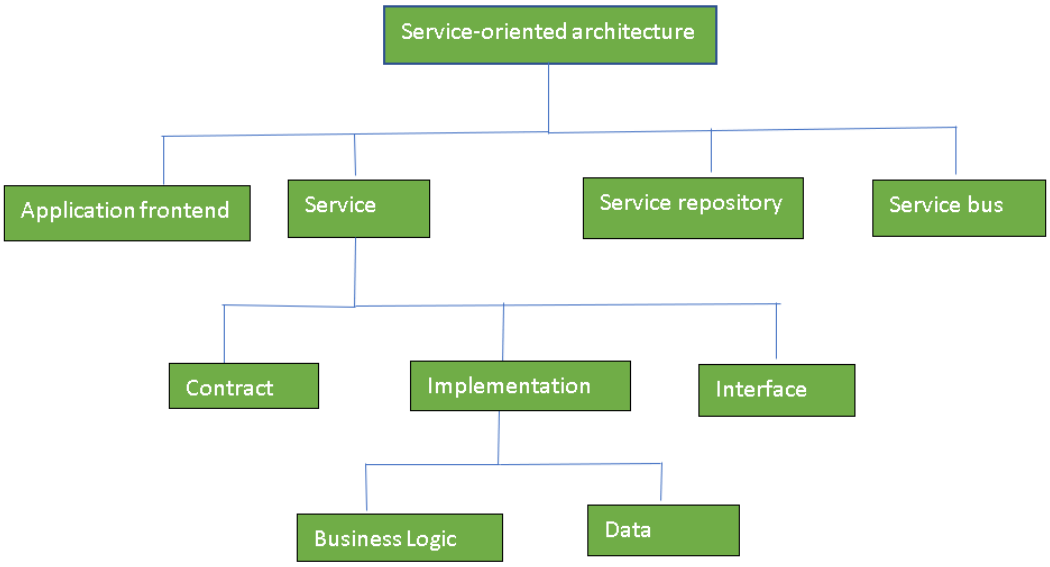
There are two major roles within Service-oriented Architecture:

1. **Service provider:** The service provider is the maintainer of the service and the organization that makes available one or more services for others to use. To advertise services, the provider can publish them in a registry, together with a service contract that specifies the nature of the service, how to use it, the requirements for the service, and the fees charged.
2. **Service consumer:** The service consumer can locate the service metadata in the registry and develop the required client components to bind and use the service.



Services might aggregate information and data retrieved from other services or create workflows of services to satisfy the request of a given service consumer. This practice is known as service orchestration Another important interaction pattern is service choreography, which is the coordinated interaction of services without a single point of control.

Components of SOA:



Guiding Principles of SOA:

1. **Standardized service contract:** Specified through one or more service description documents.
2. **Loose coupling:** Services are designed as self-contained components, maintain relationships that minimize dependencies on other services.
3. **Abstraction:** A service is completely defined by service contracts and description documents. They hide their logic, which is encapsulated within their implementation.

4. **Reusability:** Designed as components, services can be reused more effectively, thus reducing development time and the associated costs.
5. **Autonomy:** Services have control over the logic they encapsulate and, from a service consumer point of view, there is no need to know about their implementation.
6. **Discoverability:** Services are defined by description documents that constitute supplemental metadata through which they can be effectively discovered. Service discovery provides an effective means for utilizing third-party resources.
7. **Composability:** Using services as building blocks, sophisticated and complex operations can be implemented. Service orchestration and choreography provide a solid support for composing services and achieving business goals.

Advantages of SOA:

- **Service reusability:** In SOA, applications are made from existing services. Thus, services can be reused to make many applications.
- **Easy maintenance:** As services are independent of each other they can be updated and modified easily without affecting other services.
- **Platform independent:** SOA allows making a complex application by combining services picked from different sources, independent of the platform.
- **Availability:** SOA facilities are easily available to anyone on request.
- **Reliability:** SOA applications are more reliable because it is easy to debug small services rather than huge codes
- **Scalability:** Services can run on different servers within an environment, this increases scalability

Disadvantages of SOA:

- **High overhead:** A validation of input parameters of services is done whenever services interact this decreases performance as it increases load and response time.
- **High investment:** A huge initial investment is required for SOA.
- **Complex service management:** When services interact they exchange messages to tasks. the number of messages may go in millions. It becomes a cumbersome task to handle a large number of messages.

Practical applications of SOA: SOA is used in many ways around us whether it is mentioned or not.

1. SOA infrastructure is used by many armies and air forces to deploy situational awareness systems.
2. SOA is used to improve healthcare delivery.
3. Nowadays many apps are games and they use inbuilt functions to run. For example, an app might need GPS so it uses the inbuilt GPS functions of the device. This is SOA in mobile solutions.
4. SOA helps maintain museums a virtualized storage pool for their information and content.

Overview of Agile Methodology

Agile methodologies are approaches to product development that are aligned with the values and principles described in the Agile Manifesto for software development. Agile methodologies aim to deliver the right product, with incremental and frequent delivery of small chunks of functionality, through small cross-functional self-organizing teams, enabling frequent customer feedback and course correction as needed.

In doing so, Agile aims to right the challenges faced by the traditional "waterfall" approaches of delivering large products in long periods of time, during which customer requirements frequently changed, resulting in the wrong products being delivered.

Application of Agile Methodology

Through most of its brief history (since 1999-2000), "Agile" has been predominantly an approach to software development and IT application development projects. Since then, however, it now extends to other fields, too, especially in the knowledge and services industries.

Agile is about being responsive to the market and to the customer by responding quickly to their needs and demands and being able to change direction as the situation demands. Be it IT or software development or any other field where there is a flow of work and delivery of work products, Agile methods are applicable. Agile methods attempt to maximize the delivery of value to the customer and minimize the risk of building products that do not – or no longer – meet market or customer needs.

They do this by breaking up the traditionally long delivery cycle (typical of the legacy "waterfall methods") into shorter periods, called sprints or iterations. The iteration provides the cadence for delivering a working product to the customer, getting feedback and making changes based on the feedback.

Thus, Agile methods have sought to reduce delivery times (delivering early, delivering often) to ensure that smaller vertical chunks of the product get to the market, enabling customers to provide feedback early and ensure that the product they finally get meets their needs.

Agile has become an umbrella term for a variety of planning, management and technical methods and processes for managing projects, developing software and other products and services in an iterative manner. These methods include Scrum, by far the most prevalent and popular method for software, XP (eXtreme Programming or Paired Programming), and more lately Kanban.

Agile methods also include technical practices – most of which fall under the umbrella term DevOps – that enable Test Automation, Continuous Integration/ Continuous Delivery/ Deployment (CI/ CD) and overall, an ever-shrinking delivery cycle for software and other products and services.

The use of Agile as an approach to project management has increased dramatically in recent years. Gartner predicts that agile development methods will soon be used in 80% of all software development projects.

What is the Agile Manifesto?

The Agile Manifesto is a statement of core values and principles for software development. The Agile Manifesto for software development was set up in 2001 and it is a declaration of 4 vital rules and 12principles that serve as a guide for people in agile software development. It was created by 17 professionals who already practiced agile methods such as XP, DSDM, SCRUM, FDD, etc, gathered in the snowy mountains of the US state of Utah, convened by Kent Beck.

4 Core values of Agile Manifesto

Individuals and interactions over processes and tools – The first value emphasizes teamwork and communication. We must understand that software development is a human activity and that the quality of interaction between people is vital. Tools are an important part of software development, but making great software depends much more on teamwork, regardless of the tools team may use.

Working software over comprehensive documentation – Documentation has its place and can be a great resource or reference for users and coworkers alike. The main goal of software development, however, is to develop software that offers business benefits rather than extensive documentation.

Customer collaboration over contract negotiation – Development teams must work closely and communicate with their customers frequently. By listening to and getting feedback, teams will understand what all stakeholders really want.

Responding to change over following a plan – Changes are a reality in Software development, a reality that your Software process should reflect. A project plan must be flexible enough to change, as the situation demands.

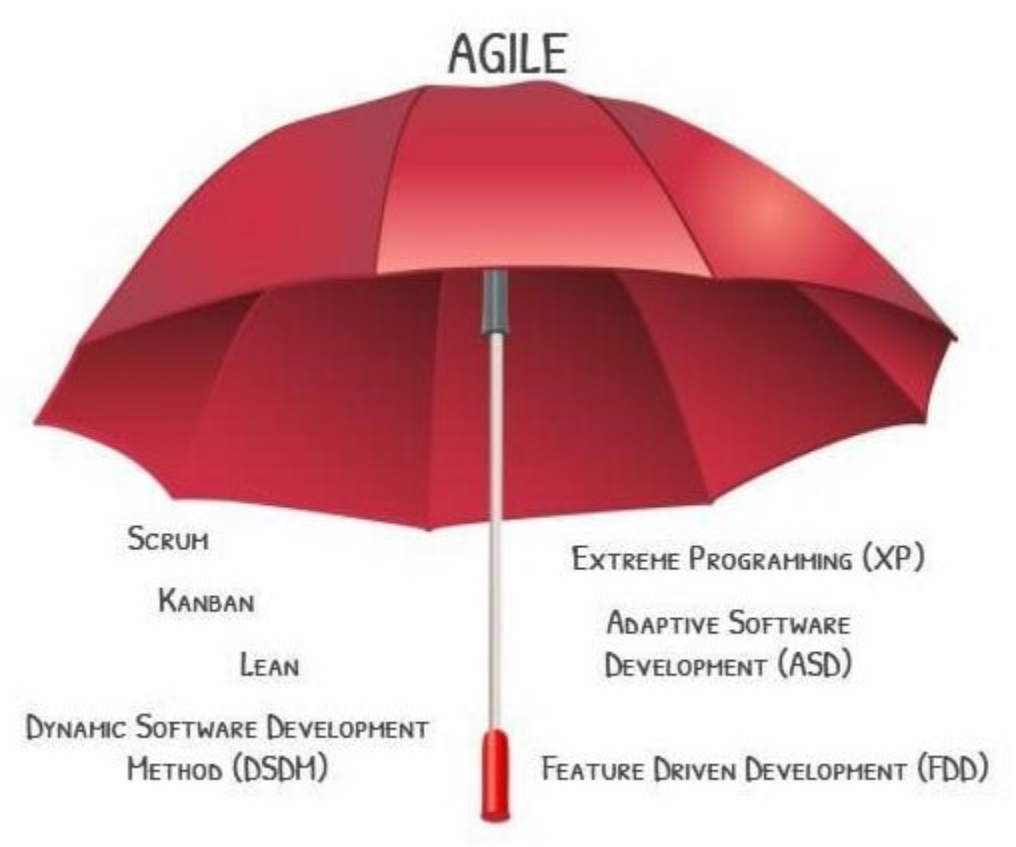
12 Principles of the Agile Manifesto

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

Key Agile Methodologies

Agile is an umbrella term for several methods and practices. Let's look at some of the popular methodologies:

- Scrum
- Extreme Programming (XP)
- Adaptive Software Development (ASD)
- Dynamic Software Development Method (DSDM)
- Feature Driven Development (FDD)
- Kanban
- Behavior Driven Development (BDD)

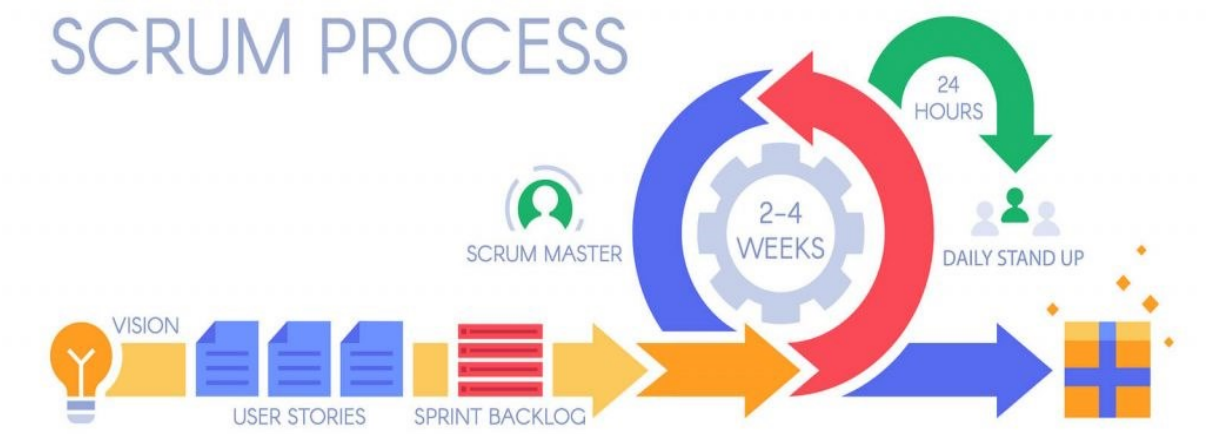


Scrum Methodology

Scrum methodology is a simple framework for working with complex projects, and it was created by Ken Schwaber and Jeff Sutherland.

Agile software development methodologies are iterative, meaning the work is divided into iterations, which are called Sprints in the case of Scrum. Scrum is executed by small teams of between 7-9 people, including a Scrum Master and a Product Owner.

In Scrum, projects are divided into cycles (typically 2 or 3 week cycles) called Sprints. The Sprint represents a timebox within which a set of features must be developed. Multiple sprints might be combined to form a Release – where formal software/ product delivery is made to the customer/ market.



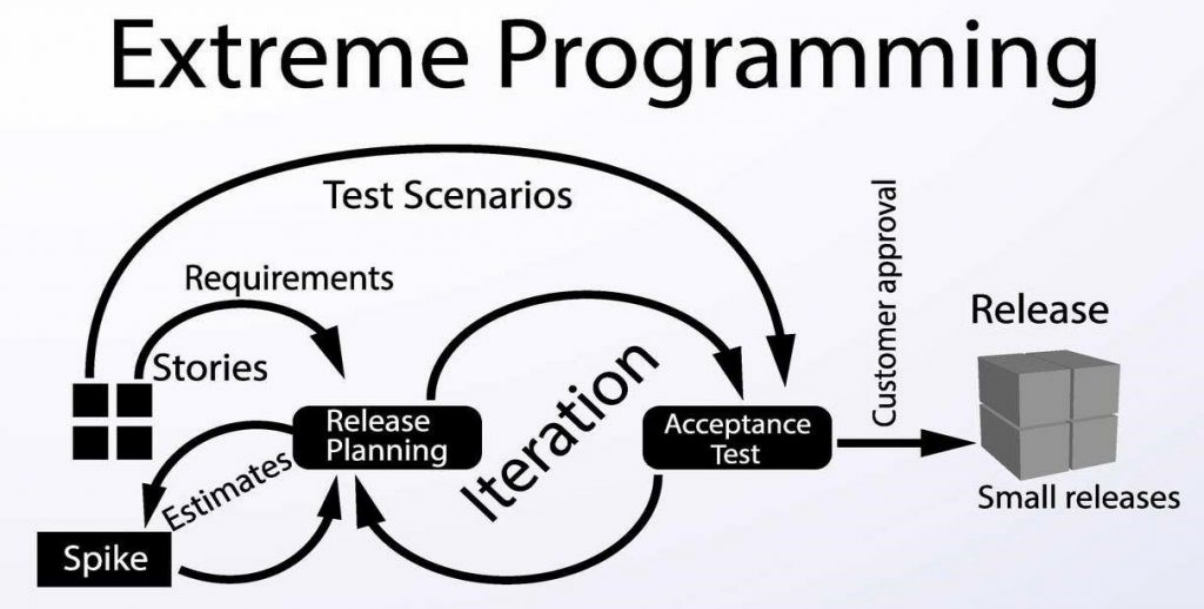
The overall product functionality is broken down by the Product Owner into smaller features (typically described as Epics and User Stories – or just Stories). These Stories are prioritized and taken up in each Sprint or Iteration. The intent of the method is for the team to be able to demo at the end of each Sprint working pieces of the product to the Product Owner, to make sure that the product is working as intended.

Overall, the Scrum method breaks the long waterfall process delivery into smaller cycles, which enables product teams and the end-customer to frequently review working software and ensure that it meets their business requirements. This ensures that the end product also meets the final requirements of the customer.

The Scrum method is characterized by specific ceremonies such as the Daily Standup meeting, the Sprint Review Meeting, the Demo to the Product Owner and the Sprint Retrospective meeting. All of these meetings provide collaboration and review opportunities to the team to ensure that development is progressing as intended, and any issues are resolved quickly.

Extreme Programming (XP)

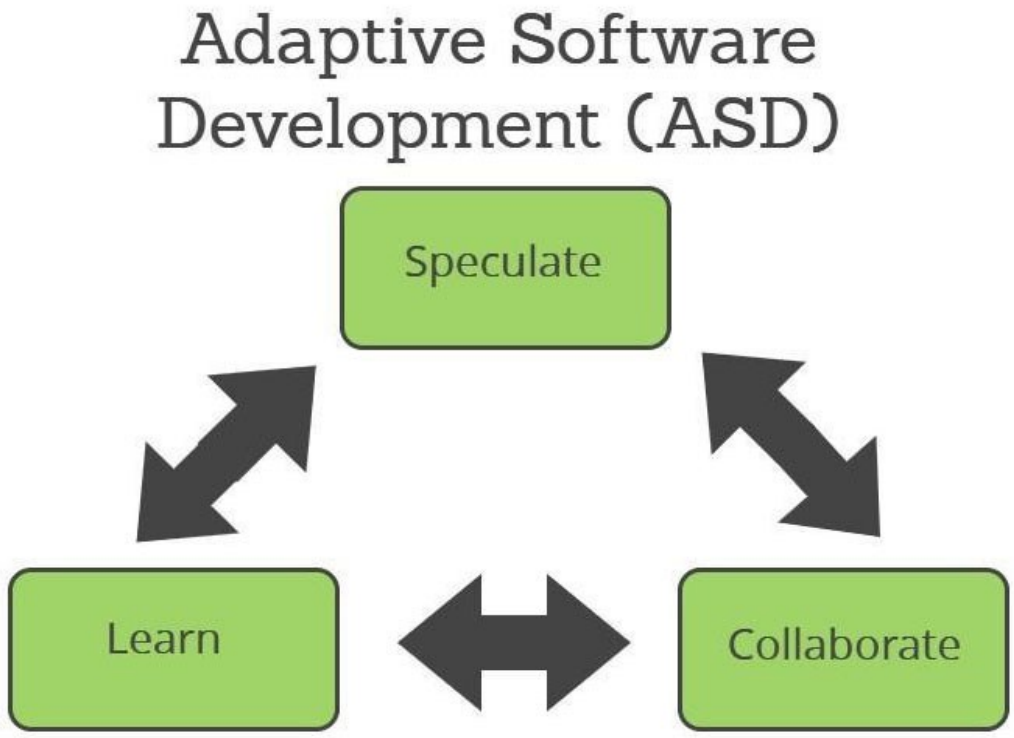
Extreme Programming (XP) – or Paired Programming is a methodology developed by Kent Beck in the early 90s. This agile methodology focuses on enhancing interpersonal relationships as a key to success in software development. XP also focuses on promoting teamwork, caring for the learning of developers, and fostering a good working environment. It is characterized by developers working in pairs where one developer programs while the other developer observes; and they switch these roles on a regular basis throughout the Sprint. This way, they enable continuous code review and feedback that enhances code quality and developer capability.



Extreme Programming (XP) promotes continuous feedback between the client and the development teams, fluid communication between all participants, simplicity in the implemented solutions and the readiness to face changes. XP is especially suitable for projects with indistinct and highly changing requirements, and where there is high technical risk.

Adaptive Software Development (ASD)

Adaptive Software Development (ASD) was developed by Jim Highsmith and Sam Bayer in the early 1990s. It incorporates the principles of continuous adaptation, i.e., *adapt to change and not fight against it*. Adaptive Software Development uses a dynamic development cycle known as Speculate, Collaborate, and Learn. This cycle is dedicated to constant learning and intense collaboration between developers and customers due to the constant change in the business environment.

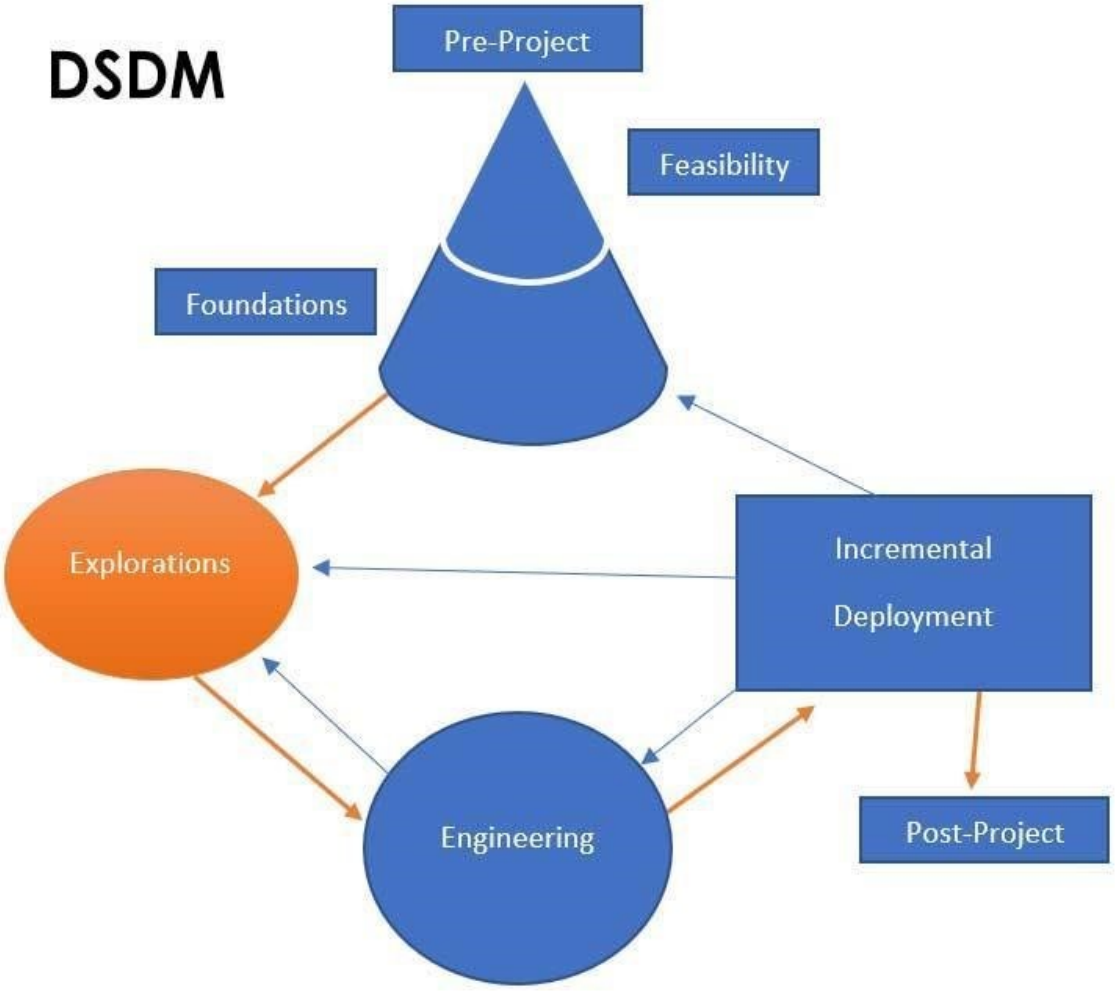


Unlike most Software development methodologies which use a static life cycle i.e., Plan-Design-Build, ASD offers a non-linear iterative life cycle, where each cycle can iterate and be modified while another cycle is being executed. It points towards Rapid Application Development (RAD), which emphasizes development speed to create a high quality, low maintenance product involving the user as much as possible. The main characteristics of ASD are:

1. Speculate: This is the initiation phase of the project where it is necessary to establish the main objectives and goals of the project by understanding the limitations (risk areas) with which the project operates.
2. Collaborate: This is the phase where most of the development is centered, maintaining co-ordination between teams that ensures what is learned by one team is communicated to the rest and does not have to be learned again by other teams from scratch.
3. Learn: The last stage ends with a series of collaboration cycles – the job is to capture what has been learned, both positive and negative. This stage is critical for the effectiveness of the project.

Dynamic Software Development Method (DSDM)

Dynamic Software Development Method (DSDM) was developed in the year 1994 by a group of vendors and experts in the field of Software development. DSDM focuses on Software projects that are characterized by tight budgets and schedules. It focuses on frequent delivery of product cycles, and development is iterative and incremental.



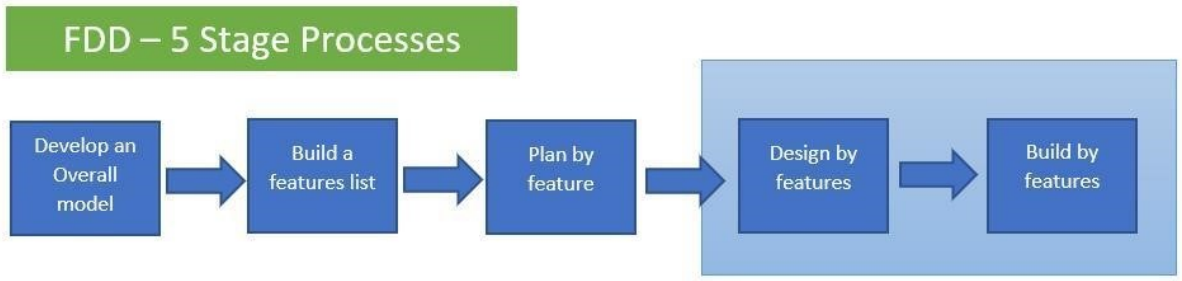
With Dynamic Software Development Method (DSDM), one can design a roadmap of early and continuous deliveries for the project, implementing an incremental solution, adapting from the feedback obtained throughout the process, and checking that the expected benefits are being met.

DSDM is an agile model that can undoubtedly help organizations that are used to working on projects to change their mentality and way of working to improve their capacity to deliver value and reduce time to market.

Feature Driven Development (FDD)

Feature Driven Development (FDD) methodology is mainly oriented for larger teams with more people than those to whom other agile methodologies such as Scrum are normally applied. FDD was developed by Jeff De Luca and Peter Coad in the year 1997. This methodology focuses on short iterations, which allow tangible deliveries of the product in a short period of time (2 weeks).

Projects with multiple teams and a large number of people represent the challenge that not all will be equally talented and disciplined. FDD includes specific activities that help address communication challenges and coordination of such projects.



FDD is a 5-stage process, the first 3 of which are sequential and the final two stages are iterative (as shown in the diagram above). All agile methodologies follow a series of principles that make them resemble each other. FDD, however, offers solutions on how to organize the team and how to program the code, which makes it especially viable for large development teams building complex software.

One of the most popular books on the FDD method was published by Stephen Palmer in 2002, titled "A Practical Guide to Feature-Driven Development".

Kanban Method

The Kanban Method was defined by David Anderson in the early -to-mid 2000s, in response to some of the challenges of the various Agile methods, especially Scrum. These methods, while trying to solve the challenges of traditional/ waterfall methods, became victim to some of the same challenges themselves.

The 2-3 week sprint cycle became too long to wait for many business contexts, the changes required in organizational structure (new roles and responsibilities) and a project management/ planning processes put too much strain on organizations, and many teams found themselves not meeting even sprint-level commitments of scope and quality. For most organizations, implementing these methods became very disruptive.

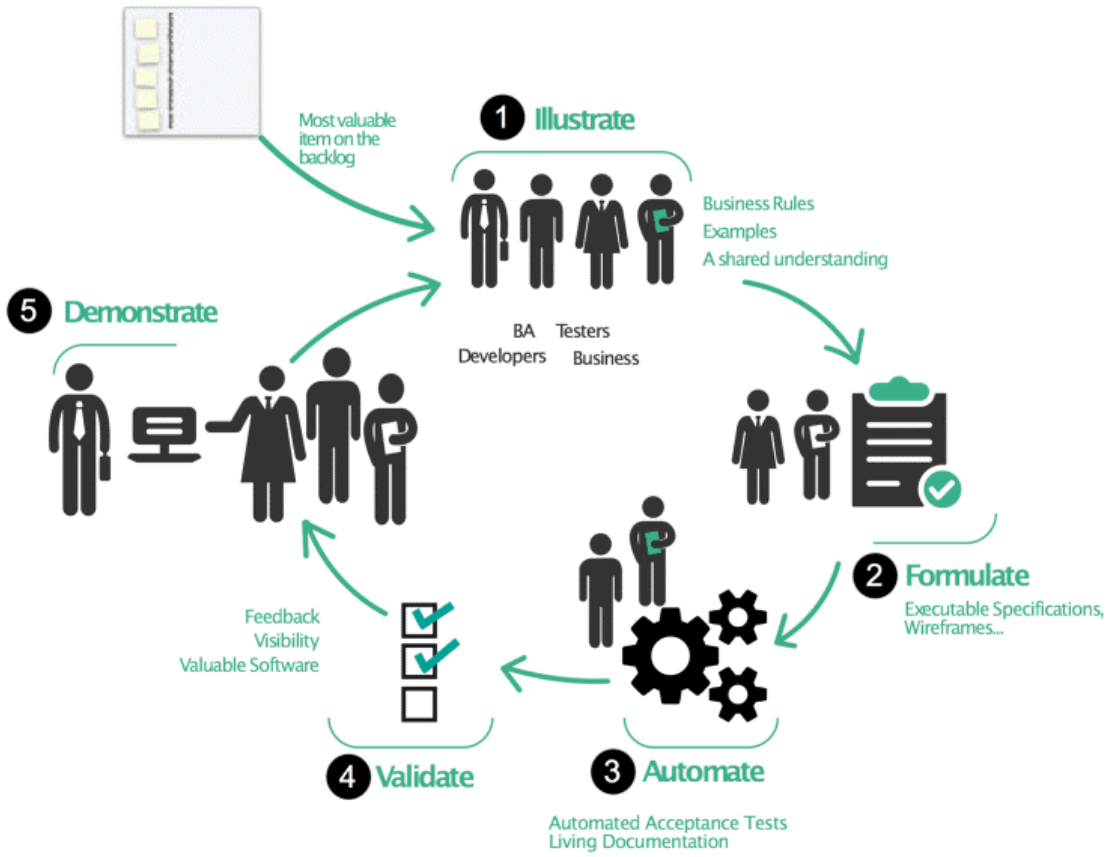
The Kanban Method was defined as the opposite of that – a non-disruptive evolutionary method for improvement, that ultimately enables teams to deliver continuously instead of in time-buckets of 2-3 weeks, get feedback faster and reduce the lead time to deliver value to the customer.

Kanban is a visual system for managing work as it moves through a process. Kanban visualizes both the process (the workflow) and the actual work passing through that process. The goal of Kanban is to identify potential bottlenecks in your process and fix them, so work can flow through it cost-effectively at an optimal speed or throughput.

Kanban is defined as a highly effective and efficient production system. The origin of the Kanban methodology lies in the "just-in-time" (JIT) production processes devised by Toyota, in which cards were used to identify material needs in the production chain. You can learn more about kanban here: https://www.digite.com/kanban/what-is-kanban/

Behavior Driven Development (BDD)

Behavior Driven Development (BDD) is a behavior-oriented agile development methodology. It was created by Dan North in 2003 as an evolution of the TDD methodology. Dan North aimed to bring non-technical people together in the process of creating the system's technical functionality. It happens that when we develop software, we involuntarily fail to include business concepts present in the functionality, resulting in a possible flow for recurring and even serious bugs.

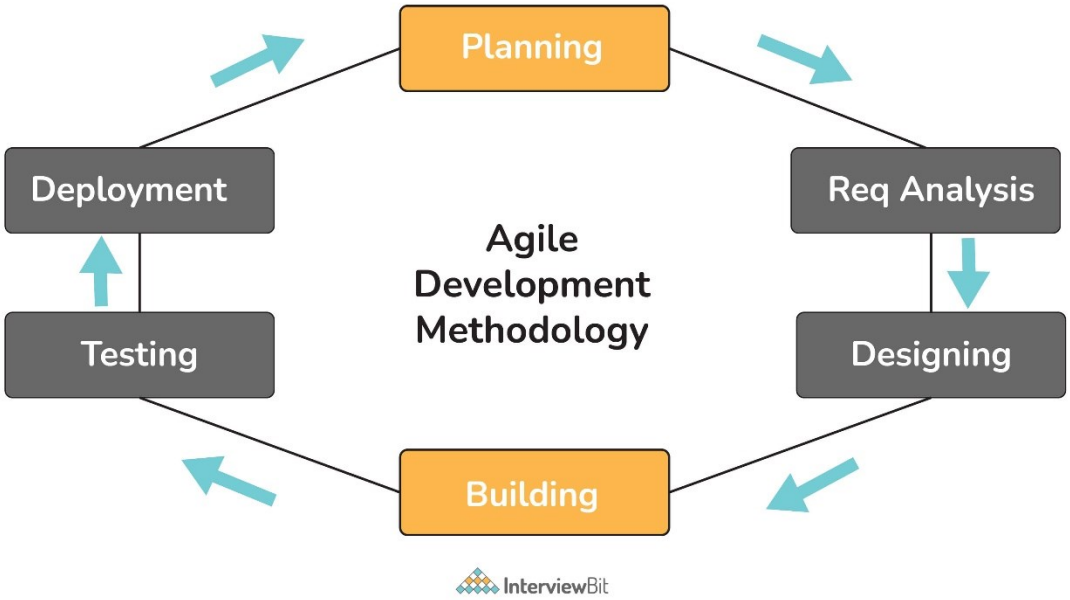


BDD uses universal language concepts that encourage collaboration between people with or without technical knowledge in a software project. The BDD development process is based on writing test scenarios and features. These contain the requirements and acceptance criteria for the system behavior. It tells you what the functionality needs to get started, what it will do next, and what the results will be after it is executed.

BDD helps teams more accurately communicate requirements, discover defects early, and build software that remains sustainable over time.

What do you mean by Agile or Agile Methodology or Agile Process?

Agile Methodology, as the name suggests, is a set of methods and practices where software development and project management take place to deliver customer-centric products in a short development cycle known as sprints. It is an iterative approach and each iteration is specially designed to be small and manageable so that it can be delivered in a specific given period of time. Agile methodologies are open to changing requirements over time and encourage constant feedback from end-users. It is the most popular approach because, in this process, customers are also involved so that they can get updates regarding their product and also make sure whether or not they are meeting their requirements.



Basic Agile Interview Questions

1. What are different types of Agile Methodology?

Different types of Agile methods or frameworks widely used in the world for software development and project development are listed below:

- **Scrum:** It is used to establish hypotheses, test them, reflect on the experience, and also make adjustments. It heavily depends on feedback, self-management, small teams, and work broken out into sprints. It relies on incremental development.
- **FDD (Feature-Driven Development):** It generally involves creating software models every two weeks and also needs development and design for each and every model feature. It is basically a lightweight iterative and incremental

software development process whose main purpose is to deliver stable and working software on time.

- **Lean Software Development:** It is basically a way of minimizing waste and maximizing value. It is more focused on process efficiency for optimum results in customer value. It is totally based on two guiding principles i.e., respect for people and continuous improvement.
- **XP (Extreme Programming):** Its main purpose is to produce higher-quality software and higher quality of life for the development team. It is considered low-risk, flexible and a way to develop software and ensures that clients get what they require. In this methodology, the software is tested right from day one, collecting feedback so as to improve the development process.
- **DSDM (Dynamic Software Development Method):** It generally focuses on the full project lifecycle and the main aim is to ensure good governance as the foundation for project management. It is user-driven and believes that modifications to the project are always expected. It also provides a full roadmap to deliver products on time and within budget.
- **ASD (Adaptive System Development):** It represents the idea that projects should always be in a state of continuous adaptation, and has a cycle of three repeating series i.e., speculate, collaborate, and learn.
- **Crystal Methodology:** It mainly focuses on individuals and their interactions rather than processes. It is considered one of the most lightweight and flexible approaches to developing software. It is a family of agile methodologies that include different variants such as crystal clear, crystal yellow, crystal orange, and crystal red.
- **Kanban:** Kanban projects are generally managed through a board or table (Kanban Board). This Kanban board is a tool that helps team members to keep an eye on workflow for measuring its progress and includes all the information that is needed to be done on the product at each stage along with its path of completion. Its main purpose is flexibility in task management, continuous improvement, and enhanced workflow.

2. What are advantages and disadvantages of Agile Process.

Advantages

There are several advantages of using the Agile Process as given below:

- Adapt well with changing requirements
- Face-to-face conversation with team members and customers
- Focuses on technical excellence and good design
- Fast and continuous development
- Enables collaboration and interaction between client and project team
- Ensure and promote customer satisfaction
- Faster feedback from customers or end-users

- Quick identification and elimination of errors found in the code
- Division of agile project into sprints or iterations i.e., short and repeatable phases typically 1-4 weeks long
- Quick delivery of products
- Easy to manage with more flexibility
- The end goal can be unknown: Agile is beneficial for projects where the goal is not defined and as the project progresses, the goal becomes more evident.

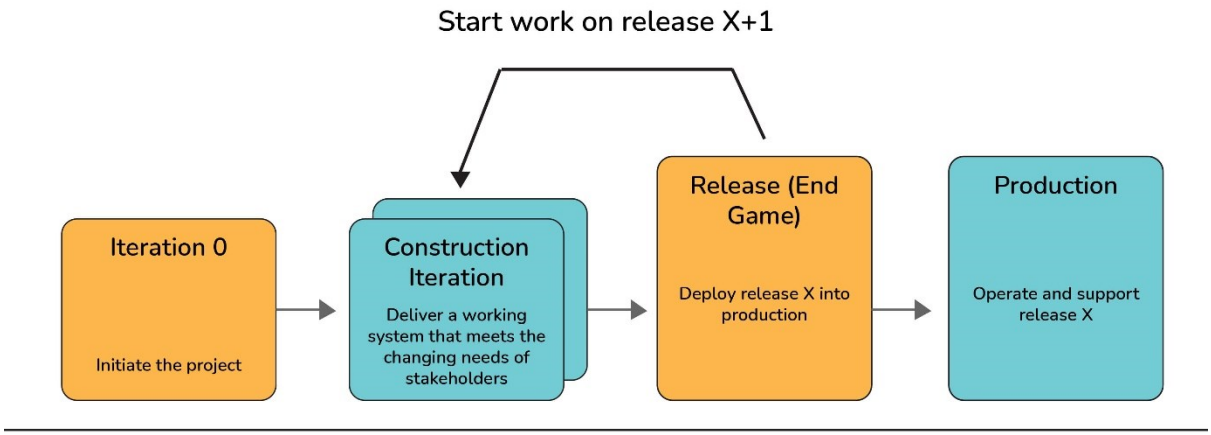
Disadvantages

There are several disadvantages of using Agile Process as given below:

- Lack of formal documentation and designing
- Difficult to estimate resource requirement and effort
- Not good for small development projects
- Costly as compared to other development methodologies
- Requires more time and energy from everyone
- Risk of ever-lasting project
- Difficult to scale large projects
- Difficulty in testing and test construction.

3. Explain Agile Testing? What are the principles of Agile Testing?

Agile testing, as the name suggests, is a software testing process where software is tested for any defects, errors, or other issues. It is considered a core part of the development process as it enables testers and developers to work together as a team that in turn improves overall performance. It also helps in ensuring the successful delivery of high-quality products. Testing is usually performed so that testers can identify and resolve the problems early and at every point in the development process.



Agile Testing Strategy

InterviewBit

Principles of Agile Testing

There are eight main principles of Agile Testing as given below:

- **Continuous Testing:** Testing should be conducted continuously by the Agile team to ensure continuous development progress.
- **Continuous Feedback:** This process generally encourages taking feedback from clients to make sure that the product meets the requirements of the client or customer.
- **Team Work or collective work:** Not only testers but developers, business analysts can also perform software testing or application testing.
- **Clean Code:** Quality of software is maintained as the team tests the software to ensure that the code is clean, simple, and tight. All errors and defects that are found during the testing phase are fixed quickly within the same iteration by the Agile Team.
- **Less Documentation:** This process usually involves the usage of reusable checklists instead of lengthy documentation.
- **Test-Driven:** In other conventional methods, testing is only performed after the implementation but in agile testing, testing is done during the implementation so that errors or any issues can be removed on time.
- **Customer Satisfaction:** During the agile testing process, development progress is being shown to clients or customers so that they can adapt and update their requirements. This is done to ensure customer satisfaction.

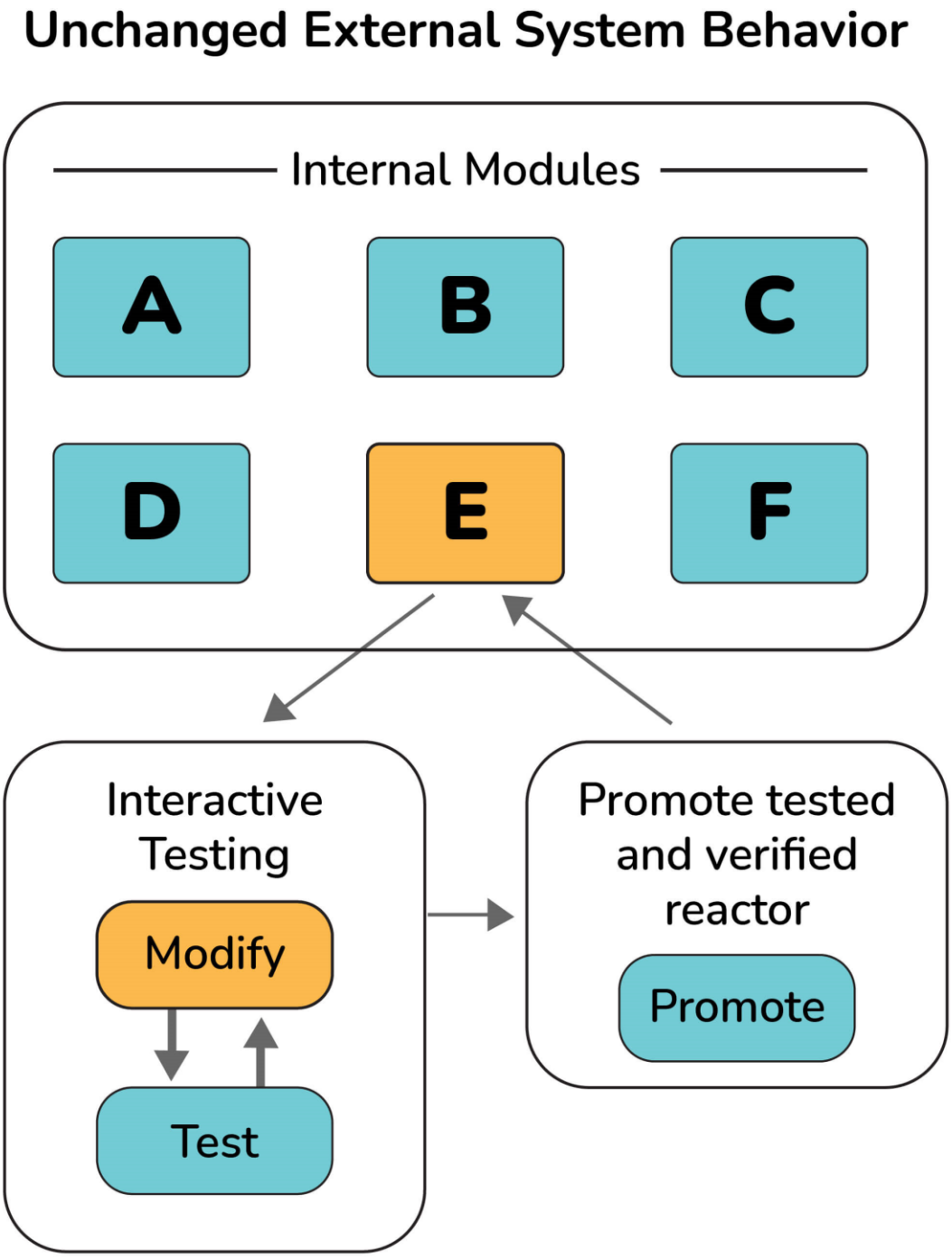
4. What good qualities an Agile Tester should have?

There are several good qualities an Agile tester should have. Some of them are listed below:

- Positive attitude and solution-oriented
- Focused towards goal
- Excellent communication skills
- Understand and fulfill customer requirements
- Basic knowledge about the Agile process and its principles
- Critical and creative thinking
- Share ideas effectively
- Plan and prioritize work on the basis of requirements
- Cope up with change

5. What do you mean by refactoring?

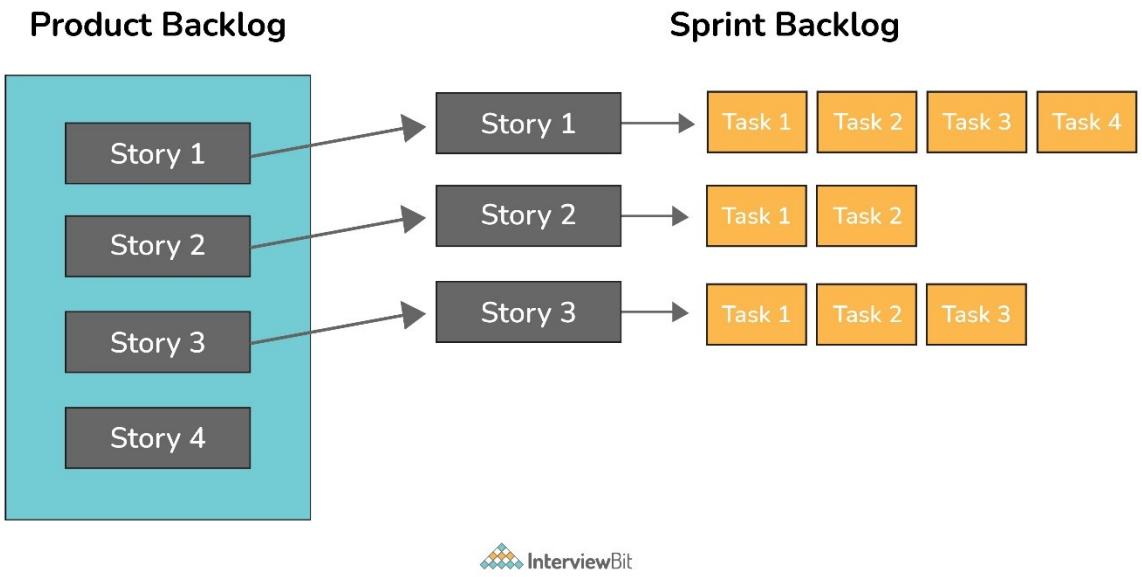
Re-factoring is basically an activity that involves alteration or modification of the internal structure of software without any change in its external behaviors or functionality. In this, developers make some changes or tinker with code to enhance and improve the internal structure of software. One of the most popular and widely used refactoring techniques in the agile software development process is Red-Green. The refactoring process makes the code more readable, understandable, and clean. The continuous habit of refactoring helps to make it easier to extend and maintain code.



Refactoring in an Isolated Environment for Change within a Larger Entity

6. What's the difference between sprint backlog and product backlog?

Sprint Backlog: It is generally owned by the development team. It only contains those features and requirements that are related to the specific sprint only. It is considered a subset of the product backlog. It is compiled of everything that must be done to complete a particular sprint. It only includes items that can be completed during each agile sprint. It is specific to the sprint goal only in a particular sprint.
Product Backlog: It is generally owned and maintained by the project owner. It usually contains each and every feature of the product as well as the requirements of the product. It is compiled to everything that must be done to complete the whole process. It just breaks down every item into a series of steps. It is more specific to the end goal of the product.



7. What is Spike and Zero Sprint in Agile?

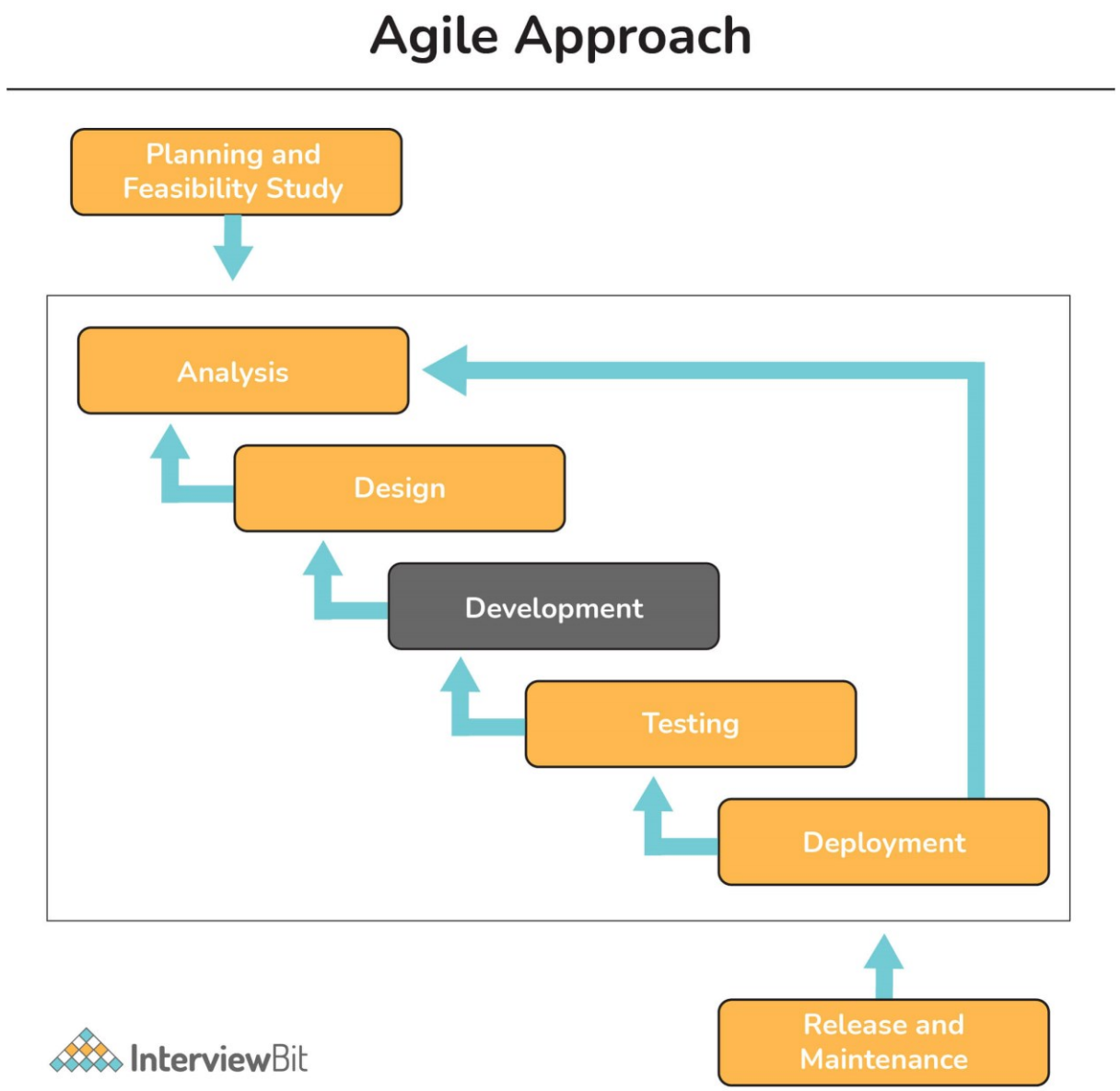
Spike: It generally refers to a too large and complex user story in software development that cannot be estimated until the development team runs a timeboxed investigation. These stories can be used for various activities like research, design, exploration, prototyping, etc. Spikes are usually created to resolve some technical issues and design problems in the project.

Zero Sprint: It generally refers to the first step or pre-preparation step that comes just before the first sprint. It includes all activities such as setting a development environment, preparing backlog, etc.

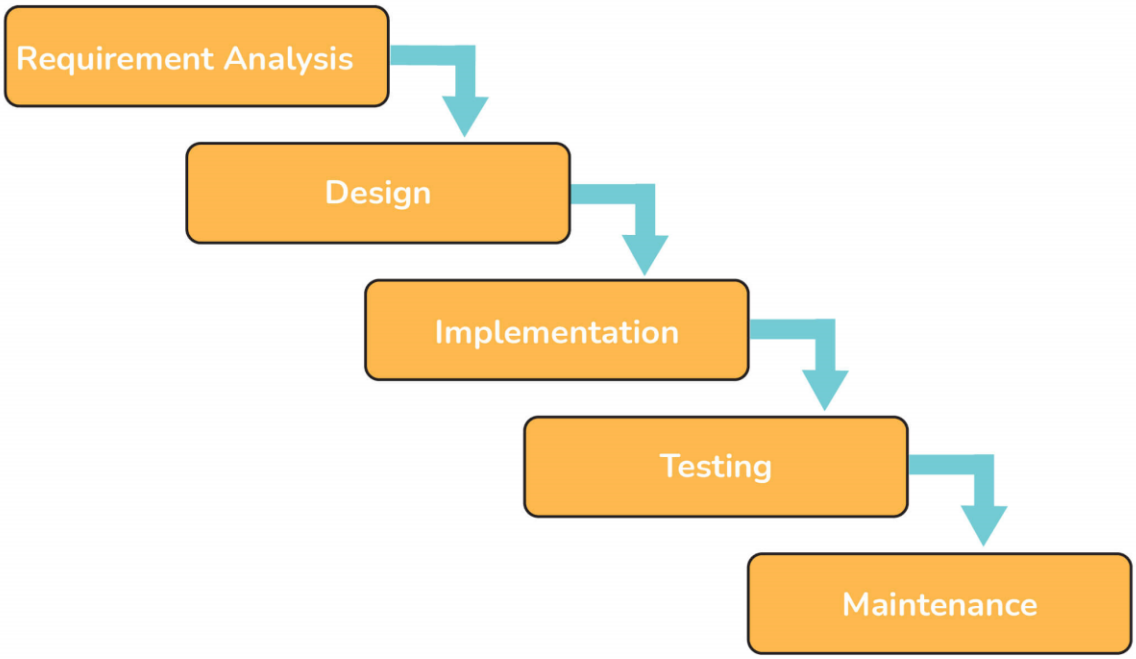
8. What's the difference between Agile methodology and Traditional methodology of Software Development?

Agile Software Development: It is an iterative approach that is used to design complicated software. In this method, project teams are allowed to be more flexible

and ensure that the final is fulfilling the customer's requirements. It develops customer-centric products and delivers in shorter sprints.



Traditional Software Development: It is a linear approach that is used to design simple software. In this method, all the phases of the process usually occur in sequence. It is more suitable for projects where the possibility of changes is negligible in the scope.

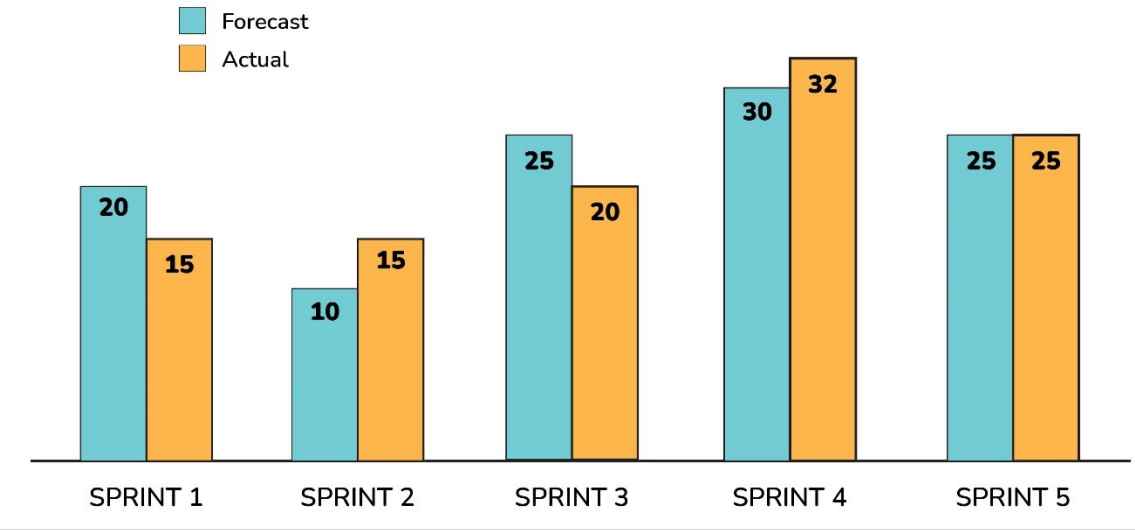


Agile Software Development	Traditional Software Development
This approach is more focused on teamwork, flexibility, customer collaboration, and features.	This approach is more focused on upfront planning and gives importance to factors like cost, scope, and time.
In this, testing is usually done parallel to the development activity.	In this, testing is usually done at the end of the development activity.
In this, testing is done on small features.	In this, testing is done on the whole application.
It involves various stakeholders including customers in the development process.	It does not involve all stakeholders in the development process.
In this methodology, testers and developers work together as a team to achieve a goal.	In this methodology, testers and developers work separately.
They collaborate with customers in each and every step throughout the process.	They collaborate with customers only at the requirement phase.

Agile Software Development	Traditional Software Development
Agile processes are more focused and flexible as compared to traditional processes.	The traditional process is less flexible as compared to the agile process.
This method is more suitable for large or more complex projects.	This method is more suitable for small or less complex projects.

9. What do you mean by the term “velocity” in Agile?

A velocity is basically a measurement unit that measures or calculates how much work an agile development team can successfully complete in a single sprint and how much time will be required to finish a project. It is widely used as a calibration tool that helps development teams to create accurate and efficient timelines. It is also used to identify problems and measure the improvements that occur with time.



InterviewBit

Velocity

10. What do you mean by Daily Stand-Up meeting?

A daily stand-up meeting is a day-to-day meeting among all the members of the agile team. Its main purpose is to know the current progress and performance of every team member that works on Scrum tasks. The meetings take place mostly in the morning and usually involves product owners, developers, and the scrum master.

These meetings usually take place for the following reasons:

- To know what was done yesterday and what is the plan for today.
- To provide a better understanding of goals.

- To make sure that every team member is working toward the same goal.
- To bring problems of team members into focus so that problems can be addressed quickly.
- To bring everyone up to date on the information and help the team to stay organized.

11. What is Incremental and Iterative Development?

Iterative Development: It is basically a software development process where software development cycles (sprint and releases) are repeated until the final product is obtained. On the basis of feedback from customers or users, the product is again developed in cycles or releases and sprints i.e., adding new functionality in a repetitive manner.
Incremental Development: It is basically a software development process where development works are sliced into increments or pieces or portions. In this, the software is developed and delivered in pieces or increments and each piece has a complete set of functionalities. The increment can either be small or large, and each increment is coded and tested fully. After testing each increment, they all are integrated so that they work as a whole.

12. What is a Product Roadmap?

A product roadmap, as the name suggests, is a powerful tool that describes how a product is likely to grow over time. It is a holistic view of product features that create the product vision. It also indicates what development is building, business goals that the new product will achieve, problems that the product will solve, etc. A product roadmap is owned by the product manager. It also encourages the development team to work together to achieve the desired goal for the successful delivery of the product.

13. What are different project management tools that are mostly used in Agile?

Different project management tools used in Agile are:

- Icescrum
- Rally Software
- Agilent
- Version One
- Agilo
- X-planner

Advanced Agile Interview Questions

14. What is the difference between Agile and Scrum?

Agile: It is an approach mainly used for software development. In this methodology, complex projects are broken down into smaller units that are achievable in a specific time frame. It always involves customers in the development process.
Scrum: There are different agile methodologies, and Scrum is one of them. It promotes accountability, function, and teamwork similar to Agile. In simple words, it is an improved way of Agile methodology and shares the same principles and values of Agile with adding some of its own unique features.

Agile vs Scrum

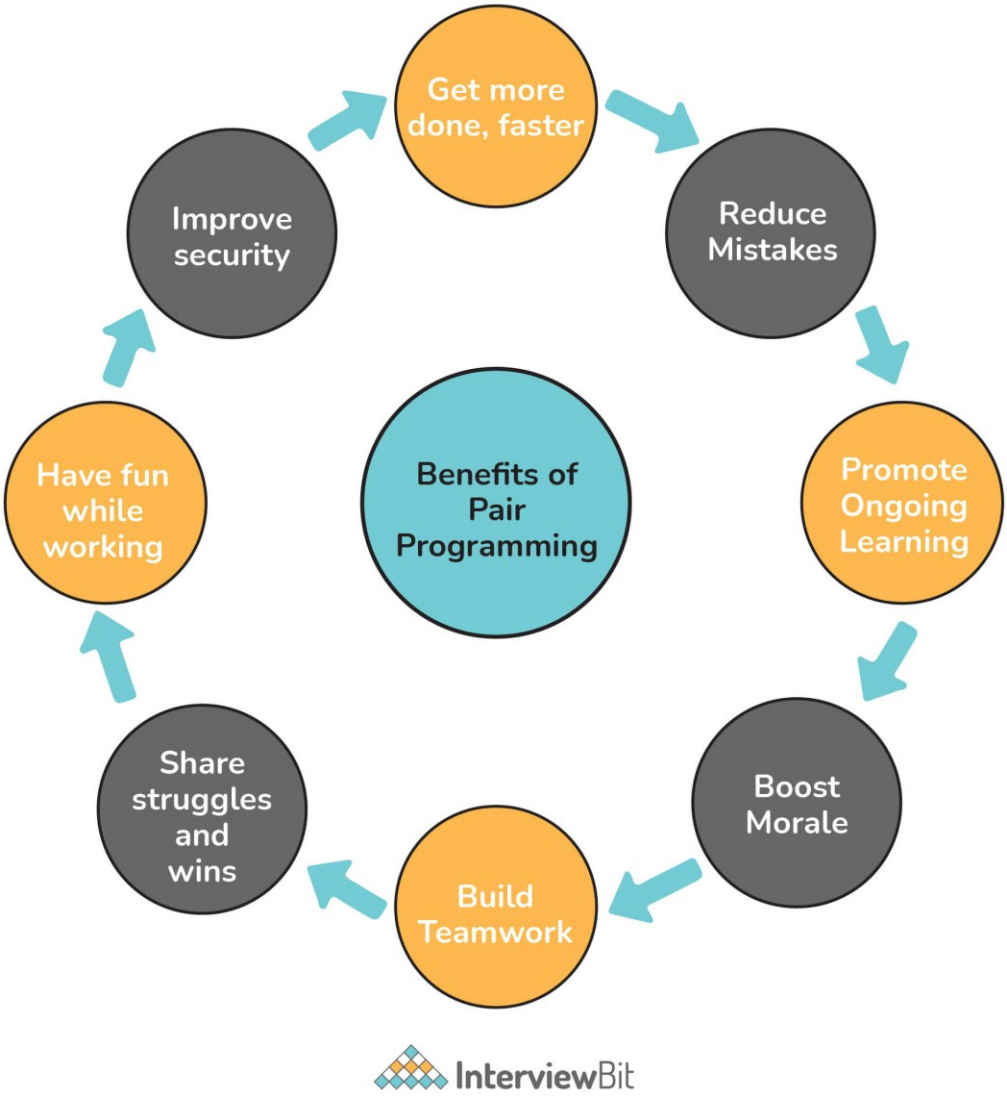
Agile and Scrum, both provide a flawless experience to customers in the software development cycle and share similar methods like collaborative iterations. But still, both of them cannot be substituted for each other. It mainly depends upon the type of project, budget, time, and feasibility to choose any one of them for project development. There are several differences between them as given below:

Agile	Scrum
It is a methodology that is used for software management and project management.	It is just a form of Agile that fully describes the process and its steps.
It emphasizes the incremental and iterative model known as sprints.	It is basically an approach or implementation of agile methodology.
It is best suited for projects that usually involve a small team of experts.	It is best suited for projects that require constant handling of changing requirements.
It is a long-term process.	It is a slow-term process.
It requires simple and straightforward design and execution.	It requires innovation, creating design, and execution.
In this, all tasks are handled and managed by the project head.	In this, all tasks and issues are addressed and handled by entire team members.
It emphasizes face-to-face communication to achieve desired goals.	It focuses on delivering maximum business value.
It is a less rigid method with more flexibility for change.	It is a more rigid method with less flexibility for change.

15. What do you mean by Pair Programming? Write its advantages.

Pair programming, as the name suggests, is a type of programming where two people write code together and work side-by-side on one machine or computer. It is basically a technique mostly used in agile software development. In this type of

programming, one person writes code and another person checks and reviews each line of code. Both of them also switch their roles while doing work.



Advantages of Pair Programming

- Develop higher-quality code
- Reduce the risk of errors
- An effective way to share knowledge
- Enhanced productivity
- Improved team collaboration

16. What is Agile Manifesto? What are its values and principles?

The agile manifesto is basically a document consisting of values and principles that are expressed in Agile. It was created in early 2001. It simply consists of 4 values and 12 key principles. This manifesto helps the development team to work more

efficiently and provides a clear and measurable structure that promotes team collaboration, iterative development, etc. It is specially designed to improve development methodologies.

The 4 Agile Values

1. **Individuals and Interactions over Processes and Tools:** It focuses on giving more attention and importance to communication with clients.
2. **Working Software over Comprehensive Documentation:** It focuses on the completion of the project and making sure that the project is completing the final deliverables.
3. **Customer Collaboration over Contract Negotiation:** It focuses on involving customers in all phases of the project so that the final product doesn't lack any requirement that the client needs. It is done to ensure 100% customer satisfaction.
4. **Responding to Change over Following a Plan:** It focuses on changes and motivates the team to adopt the change quickly so that higher quality products can be delivered. Therefore, agile works in short sprints so that changes can be utilized for good.

The 12 Agile Principles

1. **Customer Satisfaction**: First priority is to fulfill customer demands to ensure 100% customer satisfaction.
2. **Welcome Change:** Changes are important for improvement therefore even late in the development process, changes can be introduced and addressed throughout the development period.
3. **Deliver Frequently:** Products have to be delivered as soon as possible therefore focus on a shorter timescale.
4. **Work Together:** Both business stakeholders and team members work together through the development process for better collaboration.
5. **Motivated Team:** For delivering high-quality products, team members are motivated and encouraged. Team members are given the environment and support they need to perform effectively.
6. **Face-to-Face:** Agile emphasizes Face-to-face communication which is the most effective and efficient way of conveying information. It helps the team to communicate simple and complex information in an effective way.
7. **Working Software:** Delivering working software to the customer is the major concern of Agile. Working software or product is the primary measure of progress towards the final product.
8. **Constant Pace:** Agile promotes sustainable development. All teams, sponsors, developers, and users that are involved in the agile process should maintain a constant speed to deliver working software in a short timescale.
9. **Good Design:** Focuses on good design and technical details to improve quality and agility (quick and graceful).

10. **Simplicity:** Team focuses on tasks and features that are essential and reduces the amount of work and time spent on complex features and tasks that are not essential. It is done to keep things simple.
11. **Self-Organization:** Agile team should be cross-functional and self-organized. It should not depend on the manager to assign work, instead should find their own work and manage the responsibilities and timelines. Such teams not only help to deliver good quality software but also provide the best designs, requirements, and architectures.
12. **Reflect and Adjust:** To improve the effectiveness of a team, the team reflects on how to become more effective and assess their working style at regular intervals. This is done so that one can learn from their mistakes and take some steps to improve their performance in the next iterations.

17. What are Burn-up and Burn-down charts in Agile?

Burn-up Chart: It is a type of chart that is used to display or represent the amount of work that has been completed and the total amount of work for a sprint or iteration.
Burn-down Chart: It is a type of chart that is used to display or represent the amount of work that is remaining to be completed in the project. These charts are very simple and easy to understand.

18. What are different types of Burn-Down charts?

Different types of Burn-Down charts are listed below:

- **Product Burndown Chart:** It is a type of chart that is used to show story points of each completed sprint so that it depicts the completion of requirements over time. It mainly shows how many of the product goals are being achieved by the team and how much work is remaining.
- **Sprint Burndown Chart:** It is a type of chart that is used to show the remaining works for the scrum team of a particular sprint. It makes the work of the team visible and shows the rate at which work is completed and how much is remaining to be completed.
- **Release Burndown Chart:** It is a type of chart that is used to show how a team is progressing against the work for a release. This chart is updated by the scrum team at the end of each sprint. It is very essential to see what process is being made during each sprint.
- **Defect Burndown Chart:** It is a type of chart that is used to show the total number of defects that are being identified and fixed or removed.

19. Name three main Agile frameworks other than Scrum for product development.

Three main Agile Frameworks other than Scrum are:

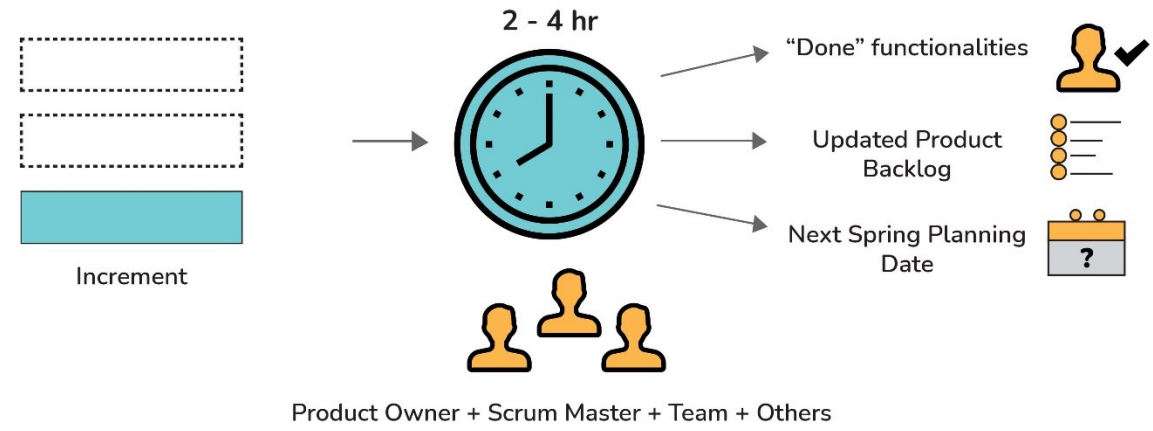
- Kanban
- Test-Driven Development (TDD)
- Feature Driven Development (FDD)

20. What is “Planning Poker” technique?

Planning Poker, also known as Scrum Poker, is a consensus-based technique that not only helps agile teams to estimate the time and effort that is required to complete each initiative on their product backlog but also identifies issues before time and within the course of a user story. It makes the meeting more short, productive and creates estimates with the involvement of the whole team. It is mainly used to avoid the influence of other participants, and force each person to think independently and give their opinion.

21. What is a Sprint Planning Meeting, Sprint Review Meeting and Sprint Retrospective Meeting?

- **Sprint Planning Meeting:** In this meeting, the discussion takes place about features and product backlog items (user stories) that are important to the team. This meeting is usually attended by the product owner, Scrum Master and Scrum Team. It is a weekly meeting and usually lasts for about an hour.
- **Sprint Review Meeting:** In this meeting, the Scrum team gives a demonstration of the product. After this, the product owner determines which items completed and which are not completed. He also adds some additional items to the product backlog on the basis of feedback from customers or stakeholders. Its main aim is to inspect the product being created in the sprint and modify it if required.



InterviewBit

Sprint Review Meeting

- **Sprint Retrospective Meeting:** This meeting takes place after the Sprint planning meeting. In this meeting, the Scrum team meets again to inspect itself and discuss the past mistakes, potential issues and methods to resolve them. Main aim of this meeting is to improve the development process. This meeting lasts for about 2-3 hours.

22. What do you mean by the term "increment"?

The increment is simply the sum or total of all the product backlog items that were completed during a sprint and the value of increments of all previous sprints. It is the total work completed within the current and previous sprints.

23. What are standard or common metrics for Agile? Explain.

Agile Metrics are basically standard metrics that are used to measure the work of the team. These metrics are used to determine the quality of work, productivity, progress, team health, etc. Its main focus is on value delivered to customers and how much end-users were impacted by it.
Standard Metrics for the Agile project

- **Velocity:** It measures the amount of work done by the development team during a sprint. It gives ideas about progress, capacity, etc.
- **Cumulative Flow Diagram:** It is a flow diagram used to measure the current status of work in progress of the team. It is simply used to track the progress of agile teams and manage flow stability.
- **Defect Removal Awareness:** It is used to measure the ability of the development team to remove defects prior to release. It helps to maintain the quality of products by a working team.
- **Work Category Allocation:** It is used to measure where we are spending or investing our time so that we can adjust our priorities.
- **Sprint Burndown Metric:** It is used to measure the total number of sprints or tasks that are completed as compared to estimated scrum tasks. It usually tracks the progress being made on tasks during a Sprint.
- **Defect Resolution Time:** It is used to measure the time taken by the team to identify and fix the defects or bugs in the software. There are several processes involved in fixing bugs.
- **Time Coverage or Code Coverage:** It is used to measure the time that is given to code during testing. It helps one to understand how much code is tested and also helps in assessing the test performance.
- **Business Value Delivered:** It is used to measure the efficiency of the working team.

Scrum Master Interview Questions

24. What is Scrum? Write its advantages.

Scrum is a lightweight process framework that helps scrum teams to work together and manage product development to deliver products in the shortest time. The product provided by the scrum team in the shortest period is known as a print. Its main aim is to manage tasks within a team-based development environment. It is especially used to manage project development for software products and can also

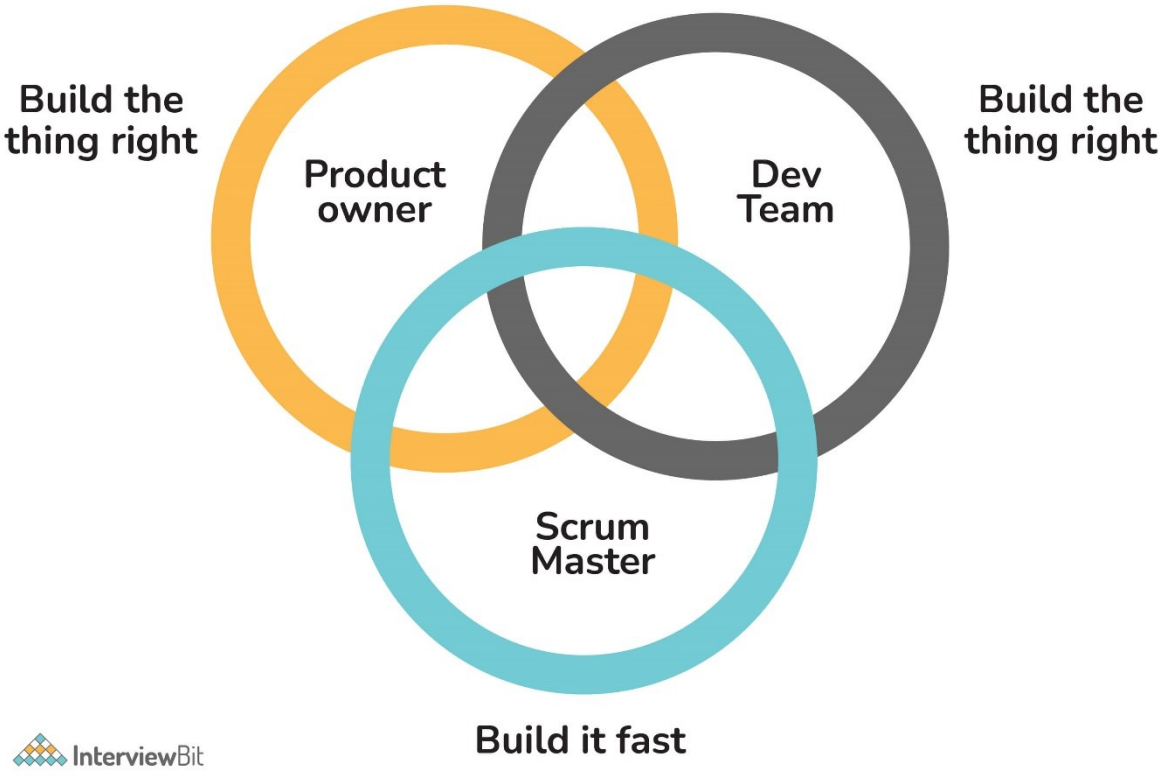
be used in business-related contexts.

Advantages of Scrum

- Releases product quickly to users and customers
- Ensures effective use of time and money and therefore saves cost
- Best suited for fast-moving development projects
- Ability to incorporates changes as they occur
- Emphasizes creativity and innovation to increase business value
- Large and complex projects are divided into small and easily manageable sprints

25. What are different roles in Scrum?

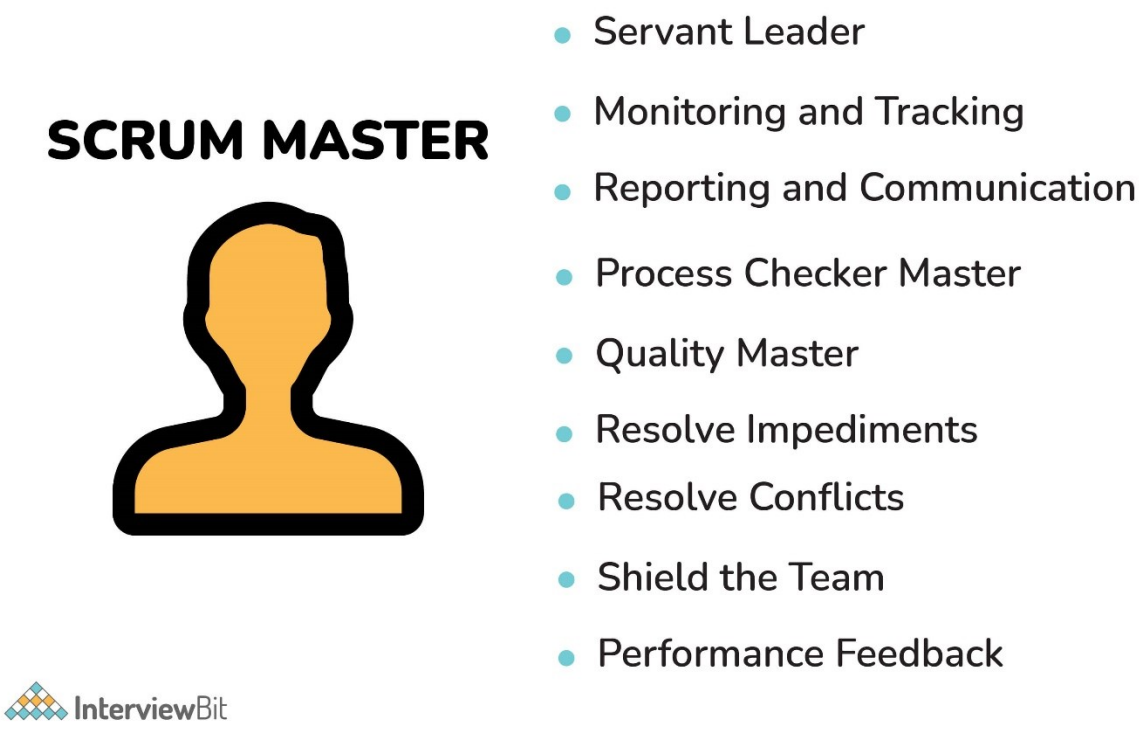
There are basically three different roles in Scrum as given below:



Scrum Master: Scrum Master is basically a team leader or supervisor of a team who is responsible for ensuring that the scrum team executes committed tasks properly.
Product Owner: The product owner is basically a stakeholder of the project who is responsible for managing the product backlog. He is also responsible for defining a vision of what to build for the team.
Development Team: It involves an individual person and each person is responsible for working collectively to complete a particular project. It is the team that is responsible for developing actual product increments and meeting sprint goals.

26. What do you mean by Scrum Master? What are the responsibilities of Scrum Master?

Scrum Master, also referred to as servant leaders, is a person who is a master of Scrum i.e., the person who is responsible for managing and facilitating an agile development team and makes sure that the scrum framework is followed. Scrum master is also referred to as coach of the team that helps team members to do and give their best as much as possible.

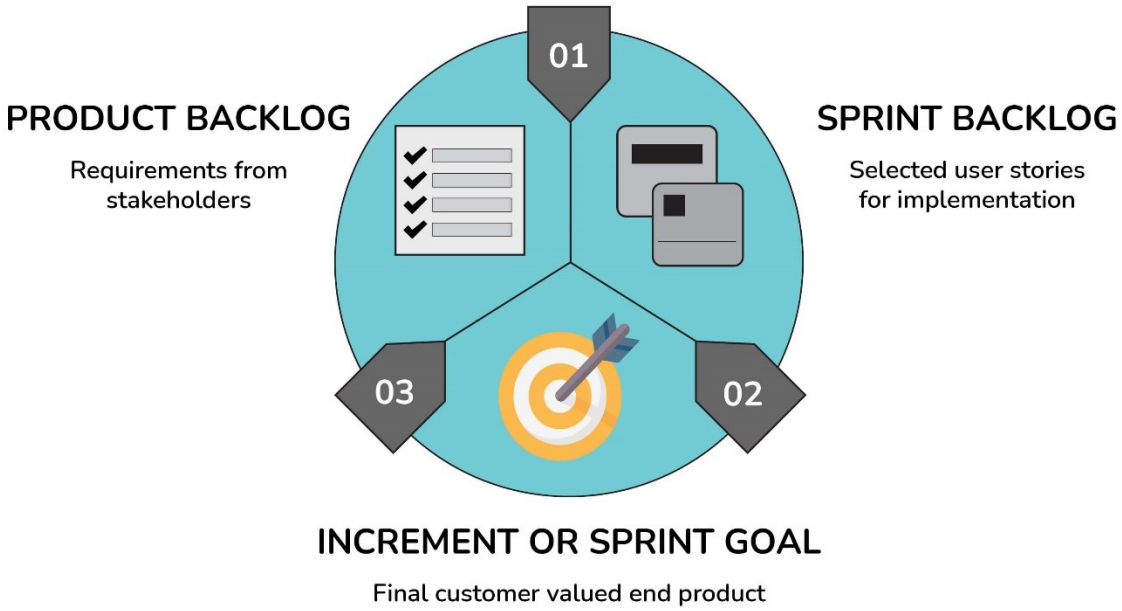


Responsibilities of Scrum Master

- Protect the team from distractions
- Motivate and guide the team to achieve the sprint goal
- Build a self-organized and motivated team
- Increase efficiency and productivity of the team
- Ensures that the team delivers expected value during the sprint
- Ensures that the team follows values, practices, and principles of Scrum
- Eliminate external blockers and manage internal roadblocks
- Lead the meetings and resolve any kind of issues

27. What are the main artifacts of Scrum Framework?

There are three main artifacts of Scrum Framework:



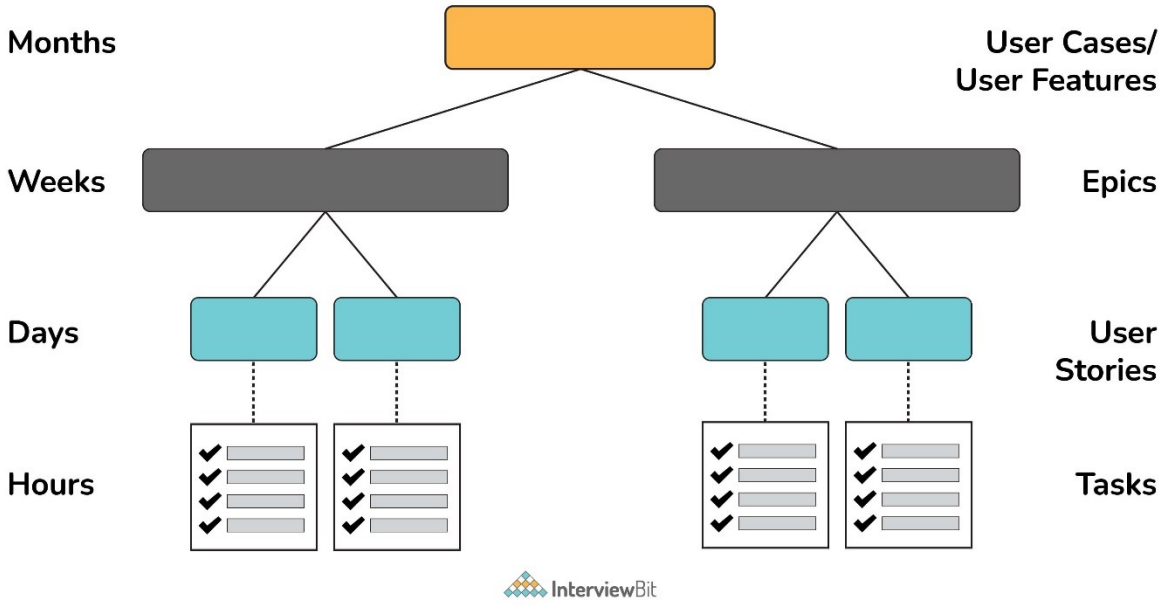
Scrum Artifacts

InterviewBit

- **Product Backlog:** It is a list of all requirements from clients or stakeholders that are needed in the product and should be accomplished before the end of the project.
- **Sprint Backlog:** It is a list of all finalized user stories, bug fixes, work items, etc., that are completed and selected by scrum to be completed during the current sprint.
- **Product Increment:** It is the version of the end product derived from the completion of each Sprint.

28. Explain the terms User story, Epic, and Tasks in Scrum?

There are a lot of technical terms that are normally used in Scrum activities. Some of them are given below:



- **Epic:** It is basically a large story that cannot be completed in a single sprint. Therefore, epics are sub-divided into multiple, smaller user stories before they can be worked on.
- **User story:** These are the smallest units that can be fitted and completed in one sprint. User stories are further broken down into different tasks.
- **Tasks:** These are detailed pieces of work that are necessary to turn user stories into workable components.

29. What are the important tools that are mostly used in a Scrum Project?

Tools mostly used in Scrum Projects are:

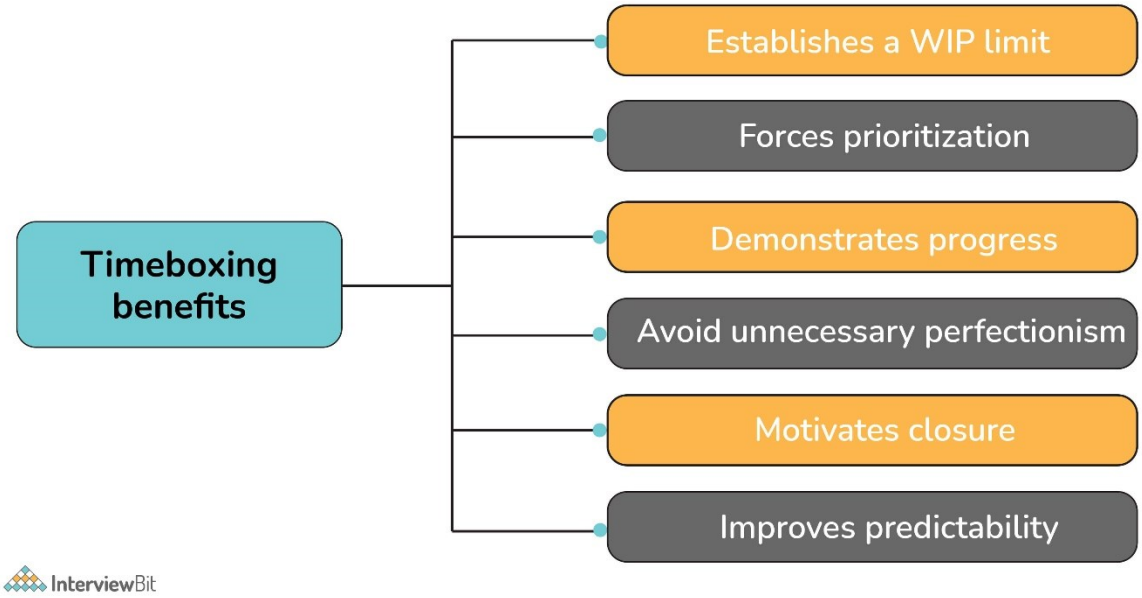
- Version One
- Sprintster
- Atlassian JIRA
- RTC Jazz, etc.

30. Explain Time Boxing in Scrum.

Timeboxing is an important time management technique or tool that is used to limit the amount of time that is being spent to complete a task. It simply allows a fixed unit of time for each and every task and this unit is known as a time box. The maximum length of the time box is 15 minutes. It not only helps to improve focus but also results in an increase in productivity. There are some events in Scrum and all these events are timeboxed which means all these events are allotted with a

maximum and fixed unit of time for the task. The events that are time-boxed are listed below:

- Sprint
- Sprint Planning
- Daily Scrum
- Sprint Review
- Sprint retrospective



31. Explain the term “impediments” in Scrum.

Impediments are something that blocks or stops the progress of teamwork. It causes the team not able to perform their task in a better way and on time that in turn also slows down the velocity. It’s the responsibility of the Scrum master to remove or resolve impediments. Impediments can be anything as listed below:

- Missing resource
- Strict boss or team member
- Technical or operational issue
- Power outage
- Lack of understanding about agile or scrum
- External issues such as war, weather, etc.
- Business problems

32. What is the main role of Sashimi in Scrum?

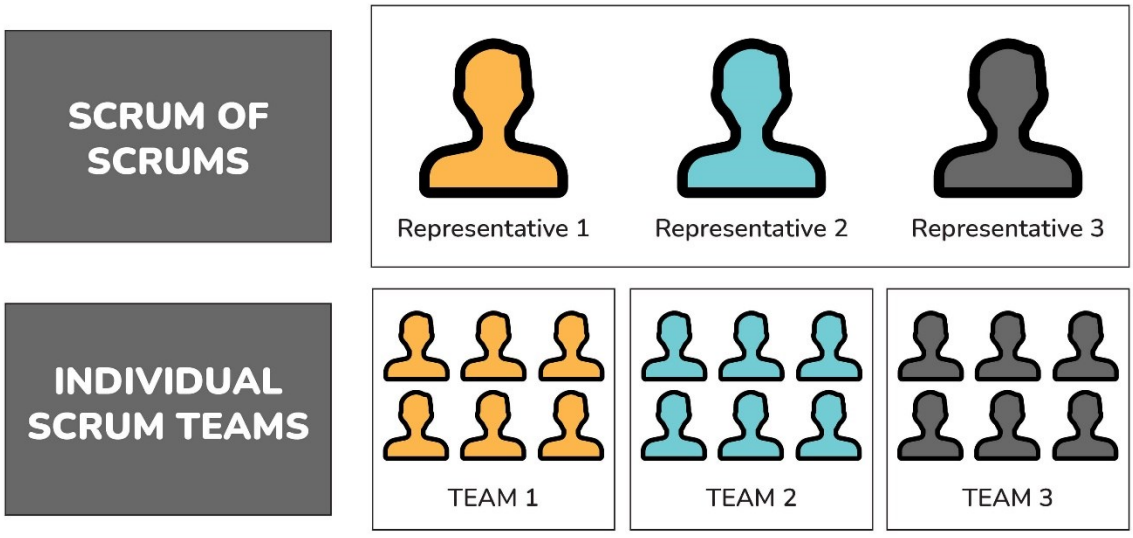
Sashimi is basically a Japanese word whose meaning is pierced body. In scrum, Sashimi is a technique that is simply used to check whether all functions (every phase of the software development cycle) are completed or not after the product is displayed. Functions include requirement analysis, planning, design, development, testing, and documentation.

33. Explain the term “story point” in Scrum.

Story point is basically a unit to estimate total efforts that are required to complete or to do a particular task or user story. It gives more accurate measures, reduces planning time, predicts releases date more accurately.

34. What do you mean by Scrum of Scrums (SoS)?

Scrum of Scrum, as the name suggests, is an Agile technique that involves meeting more than one scrum team and integrating the work of each team working on the same project. In simple words, it coordinates the work of multiple teams who need to work together to deliver complex solutions. In this meeting, members or representatives of individual teams share their high-level updates about their respective team’s work. Its main is to ensure coordination and integration of output from multiple teams by eliminating impediments if present.



InterviewBit

Scrum of Scrums Team Structure

Conclusion

35. Conclusion

After going through the above topics, you must have understood what exactly agile methodology is about and other important topics related to it. In short, agile is a process that is totally based upon flexibility, transparency, quality, and continuous improvement. It not only involves customers but also helps team members to manage work more efficiently and work effectively simply to deliver products with the highest quality within budget. All the above-mentioned questions were recently asked in Interviews and will help you to crack your interviews.