MODULE-1

Software Process Models

Software Product

Software Products are nothing but software systems delivered to the customer with the documentation that describes how to install and use the system.

In certain cases, software products may be part of system products where hardware, as well as software, is delivered to a customer. Software products are produced with the help of the software process. The software process is a way in which we produce software.

Types of software products:

Software products fall into two broad categories:

1.Generic products:

Generic products are stand-alone systems that are developed by a production unit and sold on the open market to any customer who is able to buy them.

2.Customized Products:

Customized products are the systems that are commissioned by a particular customer. Some contractor develops the software for that customer.

Essential characteristics of Well-Engineered Software Product:

A well-engineered software product should possess the following essential characteristics:

Efficiency:

The software should not make wasteful use of system resources such as memory and processor cycles.

Maintainability:

It should be possible to evolve the software to meet the changing requirements of customers.

Dependability:

It is the flexibility of the software that ought to not cause any physical or economic injury within the event of system failure. It includes a range of characteristics such as reliability, security, and safety.

In time:

Software should be developed well in time.

Within Budget:

The software development costs should not overrun and it should be within the budgetary limit.

Functionality:

The software system should exhibit the proper functionality, i.e. it should perform all the functions it is supposed to perform.

Adaptability:

The software system should have the ability to get adapted to a reasonable extent with the changing requirements memory and processor cycles.

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.

Causes of Software Crisis

1. The cost of owning and maintaining software was as expensive as developing the software

2. At that time Projects were running over-time

3. At that time Software was very inefficient

4. The quality of the software was low quality

5. Software often did not meet user requirements

6. The average software project overshoots its schedule by half

7. At that time Software was never delivered

8. Non-optimal resource utilization.

9. Difficult to alter, debug, and enhance.

10. The software complexity is harder to change.

Let’s now understand which factors are contributing to the software crisis.

1. Poor project management.

2. Lack of adequate training in software engineering.

3. Less skilled project members.

4. Low productivity improvements.

Solution of Software Crisis:

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

1. Reduction in software over budget.

2. The quality of software must be high.

3. Less time is needed for a software project.

4. Experienced and skilled people working over the software project.

5. Software must be delivered.

6. Software must meet user requirements.

Handling complexity through Abstraction and Decomposition

Abstraction:

It refers to the construction of a simpler version of a problem by ignoring the details

It is the simplification of a problem by focusing on only one aspect of the problem while omitting all other aspects. When using the principle of abstraction to understand a complex problem, we focus our attention on only one or two specific aspects of the problem and ignore the rest.

Whenever we omit some details of a problem to construct an abstraction, we construct a model of the problem. In everyday life, we use the principle of abstraction frequently to understand a problem or to assess a situation.

Decomposition:

Decomposition is a process of breaking down. It will be breaking down functions into smaller parts.

It is another important principle of software engineering to handle problem complexity. This principle is profusely made use of by several software engineering techniques to contain the exponential growth of the perceived problem complexity.

The decomposition principle is popularly is says the *divide and conquer principle.*

Functional Decomposition:

It is a term that engineers use to describe a set of steps in which they break down the overall function of a device, system, or process into its smaller parts.

Steps for the Functional Decomposition:

1. Find the most general function

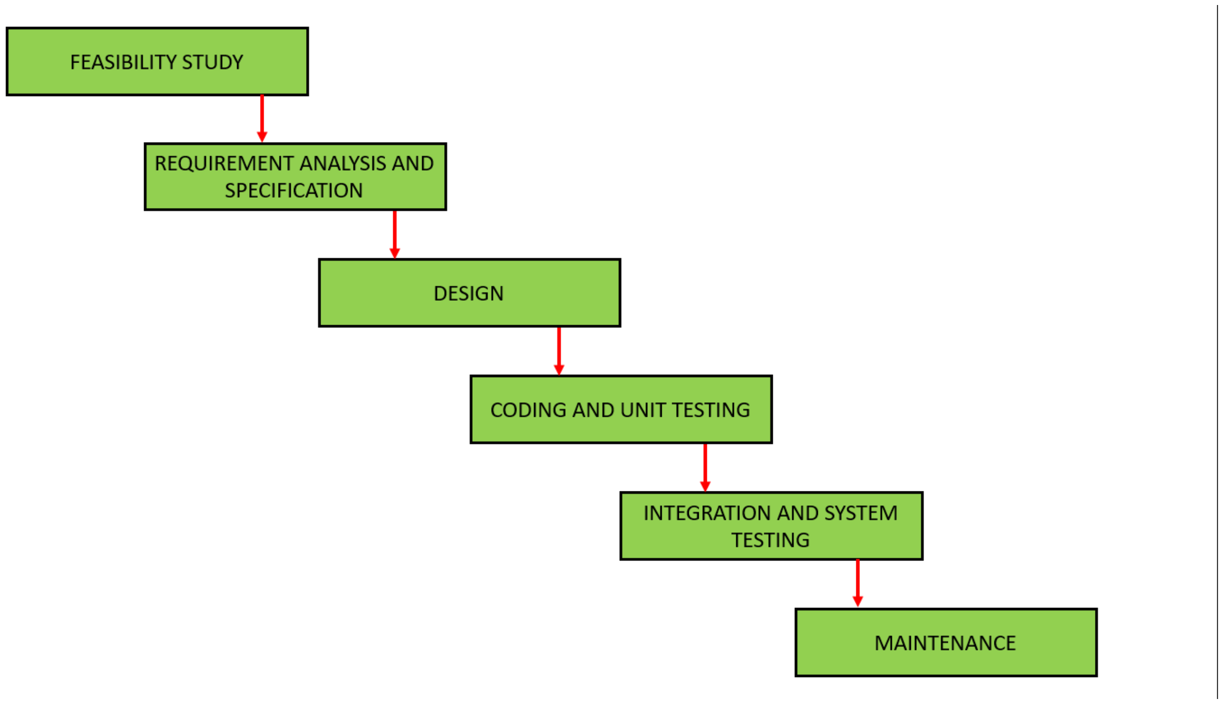
2. Find the closest sub-functions

3. Find the next levels of sub-functions

Classical Waterfall Model

The classical waterfall model is the basic **software development life cycle** model. It is very simple but idealistic. Earlier this model was very popular but nowadays it is not used. But it is very important because all the other software development life cycle models are based on the classical waterfall model.
The classical waterfall model divides the life cycle into a set of phases. This model considers that one phase can be started after the completion of the previous phase. That is the output of one phase will be the input to the next phase. Thus the development process can be considered as a sequential flow in the waterfall. Here the phases do not overlap with each other. The different sequential phases of the classical waterfall model are shown in

the below figure:


FEASIBILITY STUDY
REQUIREMENT ANALYSIS AND SPECIFICATION
DESIGN
CODING AND UNIT TESTING
INTEGRATION AND SYSTEM TESTING
MAINTENANCE

Let us now learn about each of these phases in brief detail:

1.Feasibility Study:

The main goal of this phase is to determine whether it would be financially and technically feasible to develop the software.

The feasibility study involves understanding the problem and then determining the various possible strategies to solve the problem. These different identified solutions are analyzed based on their benefits and drawbacks, The best solution is chosen and all the other phases are carried out as per this solution strategy.

2. Requirements analysis and specification:

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

▪ Requirement gathering and analysis:

Firstly all the requirements regarding the software are gathered from the customer and then the gathered requirements are analyzed. The goal of the analysis part is to remove incompleteness (an incomplete requirement is one in which some parts of the actual requirements have been omitted) and inconsistencies (an inconsistent requirement is one in which some part of the requirement contradicts some other part).

▪ **Requirement specification:**

These analyzed requirements are documented in a software requirement specification (SRS) document. SRS document serves as a contract between the development team and customers. Any future dispute between the customers and the developers can be settled by examining the SRS document.

3. Design:

The goal of this phase is to convert the requirements acquired in the SRS into a format that can be coded in a programming language. It includes high-level and detailed design as well as the overall software architecture. A Software Design Document is used to document all of this effort (SDD)

4. Coding and Unit testing:

In the coding phase software design is translated into source code using any suitable programming language. Thus each designed module is coded. The aim of the unit testing phase is to check whether each module is working properly or not.

5.Integration and System testing:

Integration of different modules are undertaken soon after they have been coded and unit tested. Integration of various modules is carried out incrementally over a number of steps. During each integration step, previously planned modules are added to the partially integrated system and the resultant system is tested. Finally, after all the modules have been successfully integrated and tested, the full working system is obtained and system testing is carried out on this.

System testing consists of three different kinds of testing activities as described below :

Alpha testing: Alpha testing is the system testing performed by the development team.

Beta testing: Beta testing is the system testing performed by a friendly set of customers.

Acceptance testing: After the software has been delivered, the customer performed acceptance testing to determine whether to accept the delivered software or reject it.

6. Maintenance:

Maintenance is the most important phase of a software life cycle. The effort spent on maintenance is 60% of the total effort spent to develop a full software. There are basically three types of maintenance :

Corrective Maintenance: This type of maintenance is carried out to correct errors that were not discovered during the product development phase.

Perfective Maintenance: This type of maintenance is carried out to enhance the functionalities of the system based on the customer's request.

Adaptive Maintenance: Adaptive maintenance is usually required for porting the software to work in a new environment such as working on a new computer platform or with a new operating system.

Advantages of Classical Waterfall Model

The classical waterfall model is an idealistic model for software development. It is very simple, so it can be considered the basis for other software development life cycle models. Below are some of the major advantages of this SDLC model:

1. This model is very simple and is easy to understand.

2. Phases in this model are processed one at a time.

3. Each stage in the model is clearly defined.

4. This model has very clear and well-understood milestones.

5. Process, actions and results are very well documented.

6. Reinforces good habits: define-before- design, design-before-code.

7. This model works well for smaller projects and projects where requirements are well understood.

Drawbacks of Classical Waterfall Model

The classical waterfall model suffers from various shortcomings, basically, we can't use it in real projects, but we use other software development lifecycle models which are based on the classical waterfall model. Below are some major drawbacks of this model:

1.No feedback path: In the classical waterfall model evolution of software from one phase to another phase is like a waterfall. It assumes that no error is ever committed by developers during any phase. Therefore, it does not incorporate any mechanism for error correction.

2.Difficult to accommodate change requests: This model assumes that all the customer requirements can be completely and correctly defined at the beginning of the project, but actually customers' requirements keep on changing with time. It is difficult to accommodate any change requests after the requirements specification phase is complete.

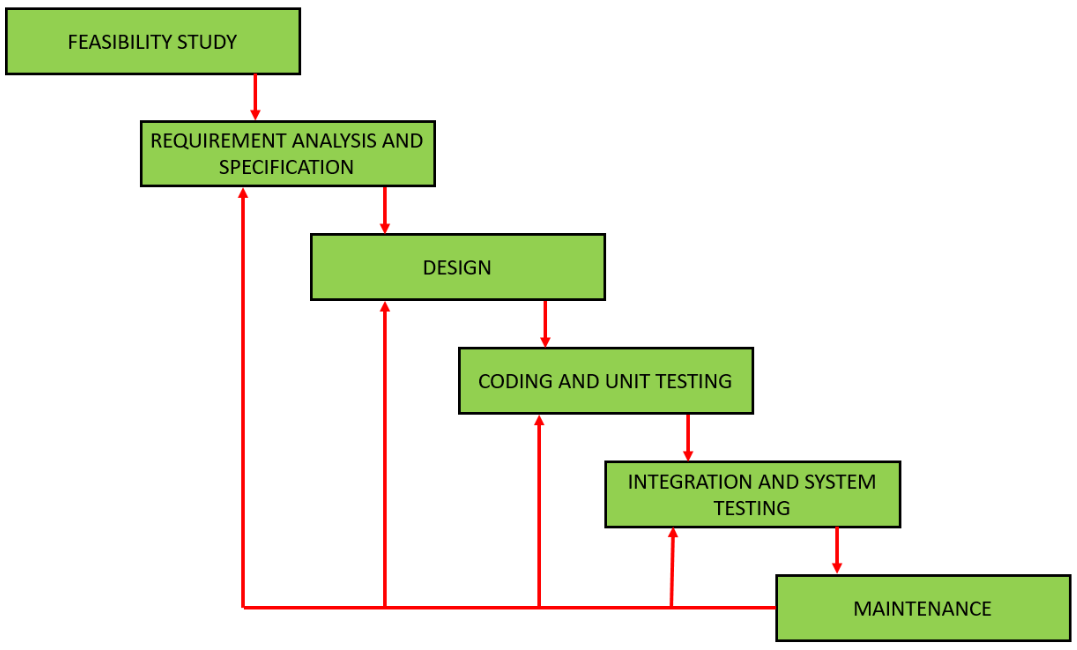
3.No overlapping of phases: This model recommends that a new phase can start only after the completion of the previous phase. But in real projects, this can't be maintained. To increase efficiency and reduce cost, phases may overlap.

Iterative Waterfall Model

In a practical software development project, the classical waterfall model is hard to use. So, the Iterative waterfall model can be thought of as incorporating the necessary changes to the classical waterfall model to make it usable in practical software development projects. It is almost the same as the classical waterfall model except some changes are made to increase the efficiency of the software development.

The iterative waterfall model provides feedback paths from every phase to its preceding phases, which is the main difference from the classical waterfall model.

Feedback paths introduced by the iterative waterfall model are shown in the figure below.



FEASIBILITY STUDY
REQUIREMENT ANALYSIS AND SPECIFICATION
DESIGN
CODING AND UNIT TESTING
INTEGRATION AND SYSTEM TESTING
MAINTENANCE

When errors are detected at some later phase, these feedback paths allow for correcting errors committed by programmers during some phase. The feedback paths allow the phase to be reworked in which errors are committed and these changes are reflected in the later phases. But, there is no feedback path to the stage – feasibility study, because once a project has been taken, does not give up the project easily.

It is good to detect errors in the same phase in which they are committed. It reduces the effort and time required to correct the errors.

The Iterative Waterfall Model is a software development approach that combines the sequential steps of the traditional Waterfall Model with the flexibility of iterative design. It allows for improvements and changes to be made at each stage of the development process, instead of waiting until the end of the project.

Real-life example: Iterative Waterfall Model could be building a new website for a small business. The process might look like this:

Requirements gathering: This is the first stage where the business owners and developers meet to discuss the goals and requirements of the website.

Design: In this stage, the developers create a preliminary design of the website based on the requirements gathered in stage 1.

Implementation: In this stage, the developers begin to build the website based on the design created in stage 2.

Testing: Once the website has been built, it is tested to ensure that it meets the requirements and functions properly.

Deployment: The website is then deployed and made live to the public.

Review and improvement: After the website has been live for a while, the business owners and developers review its performance and make any necessary improvements.

This process is repeated until the website meets the needs and goals of the business. Each iteration builds upon the previous one, allowing for continuous improvement and iteration until the final product is complete.

Phase Containment of Errors: The principle of detecting errors as close to their points of commitment as possible is known as Phase containment of errors.

Advantages of Iterative Waterfall Model

1. Feedback Path –

In the classical waterfall model, there are no feedback paths, so there is no mechanism for error correction. But in the iterative waterfall model feedback path from one phase to its preceding phase allows correcting the errors that are committed and these changes are reflected in the later phases.

2. Simple –

Iterative waterfall model is very simple to understand and use. That's why it is one of the most widely used software development models.

3. Cost-Effective –

It is highly cost-effective to change the plan or requirements in the model. Moreover, it is best suited for agile organizations.

4. Well-organized –

In this model, less time is consumed on documenting and the team can spend more time on development and designing.

Drawbacks of Iterative Waterfall Model :

1. Difficult to incorporate change requests –

The major drawback of the iterative waterfall model is that all the requirements must be clearly stated before starting the development phase. Customers may change requirements after some time but the iterative waterfall model does not leave any scope to incorporate change requests that are made after the development phase starts.

2. Incremental delivery not supported –

In the iterative waterfall model, the full software is completely developed and tested before delivery to the customer. There is no scope for any intermediate delivery. So, customers have to wait a long for getting the software.

3. Overlapping of phases not supported –

Iterative waterfall model assumes that one phase can start after completion of the previous phase, But in real projects, phases may overlap to reduce the effort and time needed to complete the project.

4. Risk handling not supported –

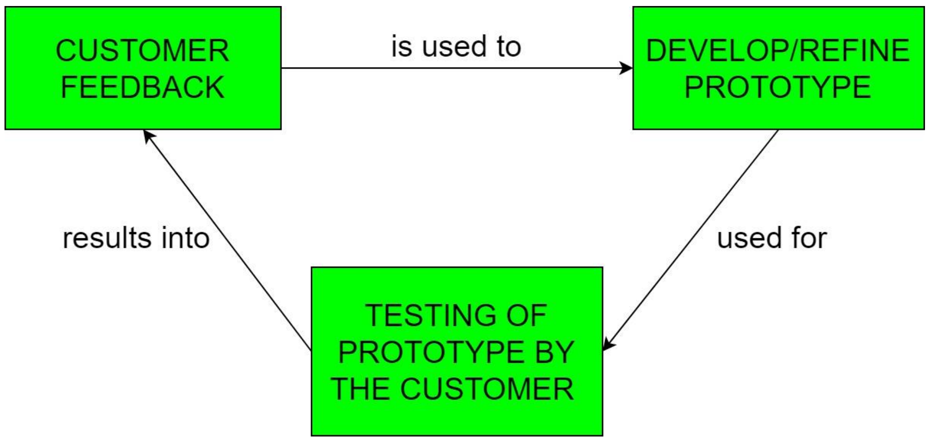
Projects may suffer from various types of risks. But, the Iterative waterfall model has no mechanism for risk handling.

5. Limited customer interactions –

Customer interaction occurs at the start of the project at the time of requirement gathering and at project completion at the time of software delivery. These fewer interactions with thecustomers may lead to many problems as the finally developed software may differ from the customers' actual requirements.

Prototyping Model

Prototyping is defined as the process of developing a working replication of a product or system that has to be engineered. It offers a small scale facsimile of the end product and is used for obtaining customer feedback as described below:



The Prototyping Model is one of the most popularly used Software Development Life Cycle Models (SDLC models). This model is used when the customers do not know the exact project requirements beforehand. In this model, a prototype of the end product is first developed, tested and refined as per customer feedback repeatedly till a final acceptable prototype is achieved which forms the basis for developing the final product.

In this process model, the system is partially implemented before or during the analysis phase thereby giving the customers an opportunity to see the product early in the life cycle. The process starts by interviewing the customers and developing the incomplete high-level paper model. This document is used to build the initial prototype supporting only the basic functionality as desired by the customer. Once the customer figures out the problems, the prototype is further refined to eliminate them. The process continues until the user approves the prototype and finds the working model to be satisfactory.

There are four types of models available:

A) Rapid Throwaway Prototyping –

This technique offers a useful method of exploring ideas and getting customer feedback for each of them. In this method, a developed prototype need not necessarily be a part of the ultimately accepted prototype. Customer feedback helps in preventing unnecessary design faults and hence, the final prototype developed is of better quality.

B) Evolutionary Prototyping –

In this method, the prototype developed initially is incrementally refined on the basis of customer feedback till it finally gets accepted. In comparison to Rapid Throwaway Prototyping, it offers a better approach which saves time as well as effort. This is because developing a prototype from scratch for every iteration of the process can sometimes be very frustrating for the developers.

C) Incremental Prototyping – In this type of incremental Prototyping, the final expected product is broken into different small pieces of prototypes and being developed individually. In the end, when all individual pieces are properly developed, then the different prototypes are collectively merged into a single final product in their predefined order. It's a very efficient approach that reduces the complexity of the development process, where the goal is divided into sub-parts and each sub-part is developed individually. The time interval between the project's beginning and final delivery is substantially reduced because all parts of the system are prototyped and tested simultaneously. Of course, there might be the possibility that the pieces just do not fit together due to some lack of ness in the development phase – this can only be fixed by careful and complete plotting of the entire system before prototyping starts.

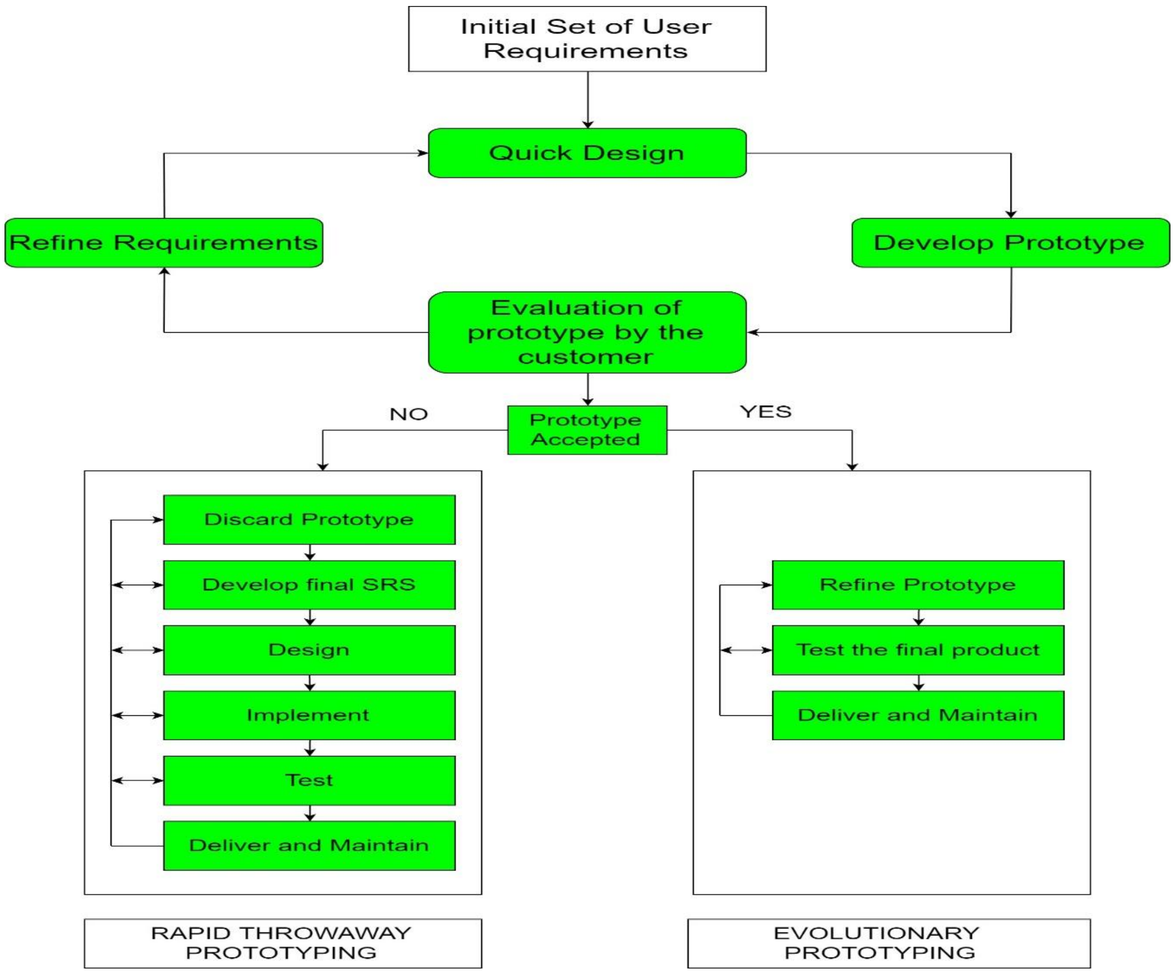
D) Extreme Prototyping – This method is mainly used for web development. It is consists of three sequential independent phases:

D.1) In this phase a basic prototype with all the existing static pages are presented in the HTML format.

D.2) In the 2nd phase, Functional screens are made with a simulated data process using a prototype services layer.

D.3) This is the final step where all the services are implemented and associated with the final prototype.

This Extreme Prototyping method makes the project cycling and delivery robust and fast, and keeps the entire developer team focus centralized on products deliveries rather than discovering all possible needs and specifications and adding unnecessitated features.



Advantages –

1. The customers get to see the partial product early in the life cycle. This ensures a greater level of customer satisfaction and comfort.

2. New requirements can be easily accommodated as there is scope for refinement.

3. Missing functionalities can be easily figured out.

4. Errors can be detected much earlier thereby saving a lot of effort and cost, besides enhancing the quality of the software.

5. The developed prototype can be reused by the developer for more complicated projects in the future.

6. Flexibility in design.

Disadvantages –

1. Costly w.r.t time as well as money.

2. There may be too much variation in requirements each time the prototype is evaluated by the customer.

3. Poor Documentation due to continuously changing customer requirements.

4. It is very difficult for developers to accommodate all the changes demanded by the customer.

5. There is uncertainty in determining the number of iterations that would be required before the prototype is finally accepted by the customer.

6. After seeing an early prototype, the customers sometimes demand the actual product to be delivered soon.

7. Developers in a hurry to build prototypes may end up with sub-optimal solutions.

8. The customer might lose interest in the product if he/she is not satisfied with the initial prototype.

Use –

The Prototyping Model should be used when the requirements of the product are not clearly understood or are unstable. It can also be used if requirements are changing quickly. This model can be successfully used for developing user interfaces, high technology software-intensive systems, and systems with complex algorithms and interfaces. It is also a very good choice to demonstrate the technical feasibility of the product.

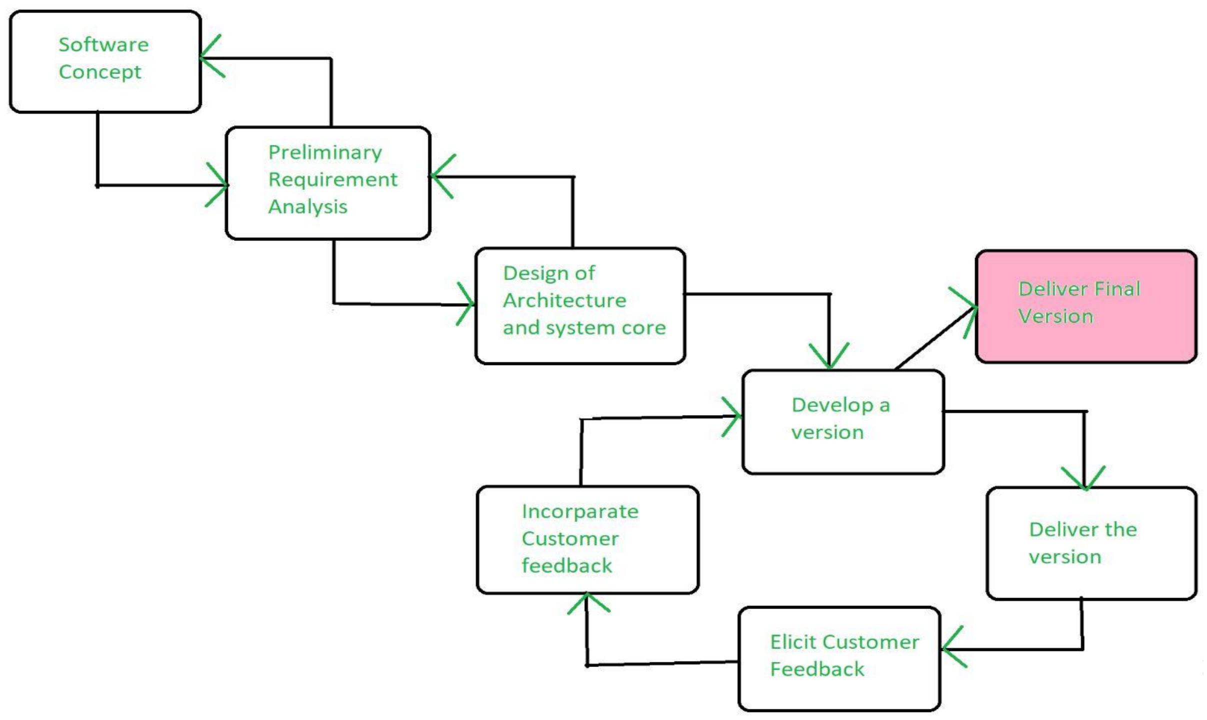
Evolutionary Model

Evolutionary model is a combination of Iterative and Incremental model of software development life cycle.

It is better for software products that have their feature sets redefined during development because of user feedback and other factors.

The Evolutionary development model divides the development cycle into smaller, incremental waterfall models in which users are able to get access to the product at the end of each cycle.

Feedback is provided by the users on the product for the planning stage of the next cycle and the development team responds, often by changing the product, plan or process. Therefore, the software product evolves with time.


Software Concept
Preliminary Requirement Analysis
Design of Architecture and system core
Deliver Final Version
Develop a version
Incorparate Customer feedback
Deliver the version
Elicit Customer Feedback

Evolutionary model suggests breaking down of work into smaller chunks, prioritizing them and then delivering those chunks to the customer one by one. The number of chunks is huge and is the number of deliveries made to the customer.

Application of Evolutionary Model:

1. It is used in large projects where you can easily find modules for incremental implementation. Evolutionary model is commonly used when the customer wants to start using the core features instead of waiting for the full software.

2. Evolutionary model is also used in object oriented software development because the system can be easily portioned into units in terms of objects.

Necessary conditions for implementing this model:-

1. Customer needs are clear and been explained in deep to the developer team.

2. There might be small changes required in separate parts but not a major change.

3. As it requires time, so there must be some time left for the market constraints.

4. Risk is high and continuous targets to achieve and report to customer repeatedly.

5. It is used when working on a technology is new and requires time to learn.

Advantages:

1. In evolutionary model, a user gets a chance to experiment partially developed system.

2. It reduces the Sometimes it is hard to divide the problem into several versions that would be acceptable to the customer which can be incrementally implemented and delivered.

Disadvantages:

Sometimes it is hard to divide the problem into several versions that would be acceptable to the customer which can be incrementally implemented and delivered**.**

Spiral Model

The spiral model, initially proposed by Boehm, is an evolutionary software process model that couples the iterative feature of prototyping with the controlled and systematic aspects of the linear sequential model. It implements the potential for rapid development of new versions of the software. Using the spiral model, the software is developed in a series of incremental releases. During the early iterations, the additional release may be a paper model or prototype. During later iterations, more and more complete versions of the engineered system are produced**.**

The Spiral Model is shown in fig:

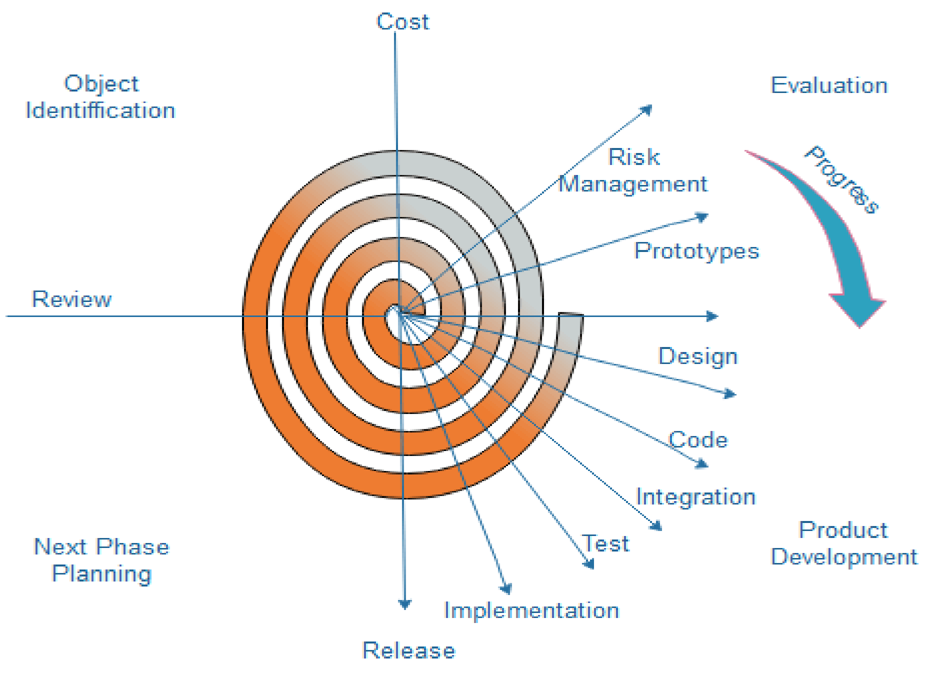


Fig. Spiral Model

Each cycle in the spiral is divided into four parts:

1. Objective setting: Each cycle in the spiral starts with the identification of purpose for that cycle, the various alternatives that are possible for achieving the targets, and the constraints that exists.

2. Risk Assessment and reduction: The next phase in the cycle is to calculate these various alternatives based on the goals and constraints. The focus of evaluation in this stage is located on the risk perception for the project.

3. Development and validation: The next phase is to develop strategies that resolve uncertainties and risks. This process may include activities such as benchmarking, simulation, and prototyping.

4. Planning: Finally, the next step is planned. The project is reviewed, and a choice made whether to continue with a further period of the spiral. If it is determined to keep, plans are drawn up for the next step of the project.

The development phase depends on the remaining risks. For example, if performance or user-interface risks are treated more essential than the program development risks, the next phase may be an evolutionary development that includes developing a more detailed prototype for solving the risks.

The risk-driven feature of the spiral model allows it to accommodate any mixture of a specification-oriented, prototype-oriented, simulation-oriented, or another type of approach. An essential element of the model is that each period of the spiral is completed by a review that includes all the products developed during that cycle, including plans for the next cycle. The spiral model works for development as well as enhancement projects.

When to use Spiral Model?

1. When deliverance is required to be frequent.

2. When the project is large

3, When requirements are unclear and complex

4. When changes may require at any time

5. Large and high budget projects

Advantages

1. High amount of risk analysis

2. Useful for large and mission-critical projects.

Disadvantages

1. Can be a costly model to use.

2. Risk analysis needed highly particular expertise

3. Doesn't work well for smaller projects.

RAD (Rapid Application Development) Model

RAD is a linear sequential software development process model that emphasizes a concise development cycle using an element based construction approach. If the requirements are well understood and described, and the project scope is a constraint, the RAD process enables a development team to create a fully functional system within a concise time period.

RAD (Rapid Application Development) is a concept that products can be developed faster and of higher quality through:

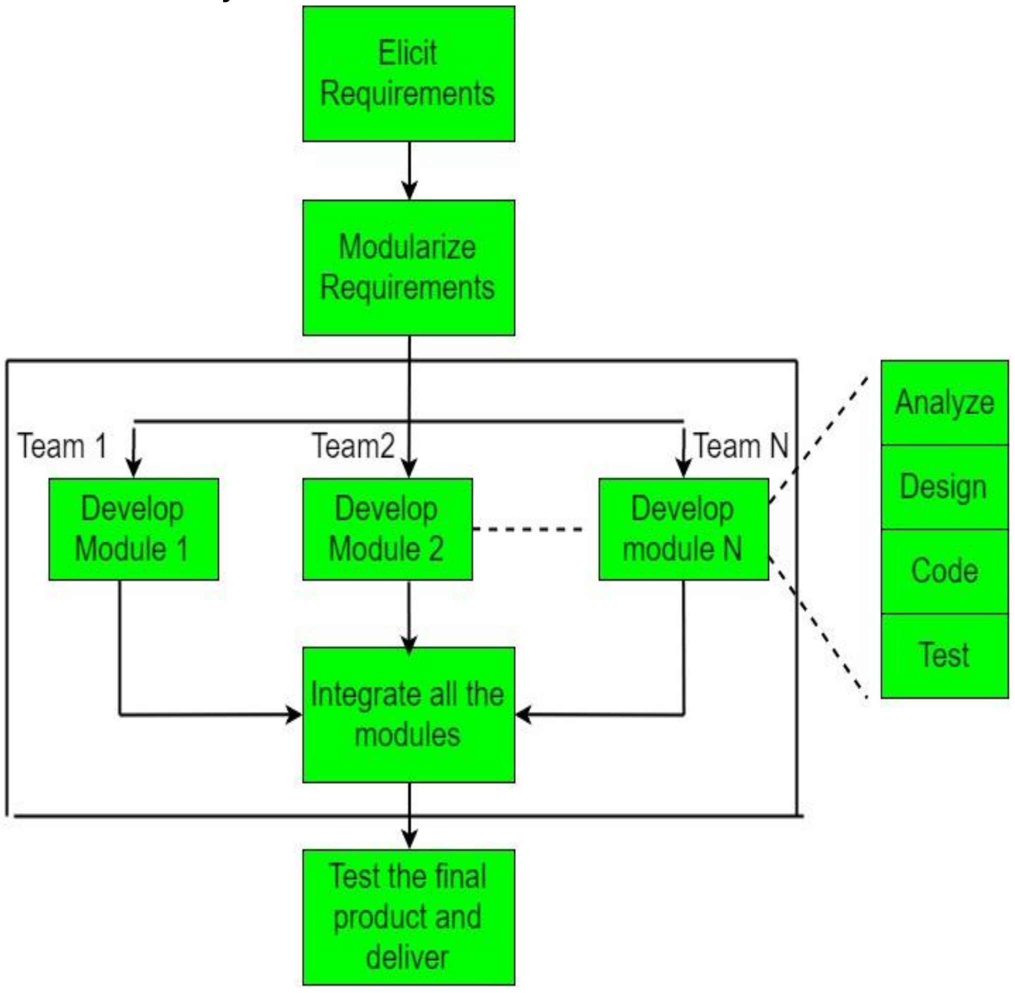
1. Gathering requirements using workshops or focus groups

2. Prototyping and early, reiterative user testing of designs

3. The re-use of software components

4. A rigidly paced schedule that refers design improvements to the next product version

5. Less formality in reviews and other team communication



The various phases of RAD are as follows:

1.Business Modelling: The information flow among business functions is defined by answering questions like what data drives the business process, what data is generated, who generates it, where does the information go, who process it and so on.

2. Data Modelling: The data from business modeling is refined into a set of data objects (entities) that are needed to support the business. The attributes (character of each entity) are identified, and the relation between these data objects (entities) is defined.

3. Process Modelling: The information object defined in the data modeling phase are transformed to achieve the data flow necessary to implement a business function. Processing descriptions are created for adding, modifying, deleting, or retrieving a data object.

4. Application Generation: Automated tools are used to facilitate construction of the software; even they use the 4th GL techniques.

5. Testing & Turnover: Many of the programming components have already been tested since RAD emphasis reuse. This reduces the overall testing time. But the new part must be tested, and all interfaces must be fully exercised.

When to use RAD Model?

1. When the system should need to create the project that modularizes in a short span time (2-3 months).

2. When the requirements are well-known.

3. When the technical risk is limited.

4. When there's a necessity to make a system, which modularized in 2-3 months of period.

5. It should be used only if the budget allows the use of automatic code generating tools.

Advantage of RAD Model

1. This model is flexible for change.

2. In this model, changes are adoptable.

3. Each phase in RAD brings highest priority functionality to the customer.

4. It reduced development time.

5. It increases the reusability of features.

Disadvantage of RAD Model

1. It required highly skilled designers.

2. All application is not compatible with RAD.

3. For smaller projects, we cannot use the RAD model.

4. On the high technical risk, it's not suitable.

5. Required user involvement.

Agile Model

The meaning of Agile is swift or versatile.Agile process model" refers to a software development approach based on iterative development**.**

Agile methods break tasks into smaller iterations, or parts do not directly involve long term planning.

The project scope and requirements are laid down at the beginning of the development process. Plans regarding the number of iterations, the duration and the scope of each iteration are clearly defined in advance.

Each iteration is considered as a short time "frame" in the Agile process model, which typically lasts from one to four weeks.

The division of the entire project into smaller parts helps to minimize the project risk and to reduce the overall project delivery time requirements. Each iteration involves a team working through a full software development life cycle including planning, requirements analysis, design, coding, and testing before a working product is demonstrated to the client.

Phases of Agile Model:

Following are the phases in the Agile model are as follows:

1. Requirements gathering

2. Design the requirements

3. Construction/ iteration

4. Testing/ Quality assurance

5. Deployment

6. Feedback

1. Requirements gathering: In this phase, you must define the requirements. You should explain business opportunities and plan the time and effort needed to build the project. Based on this information, you can evaluate technical and economic feasibility.

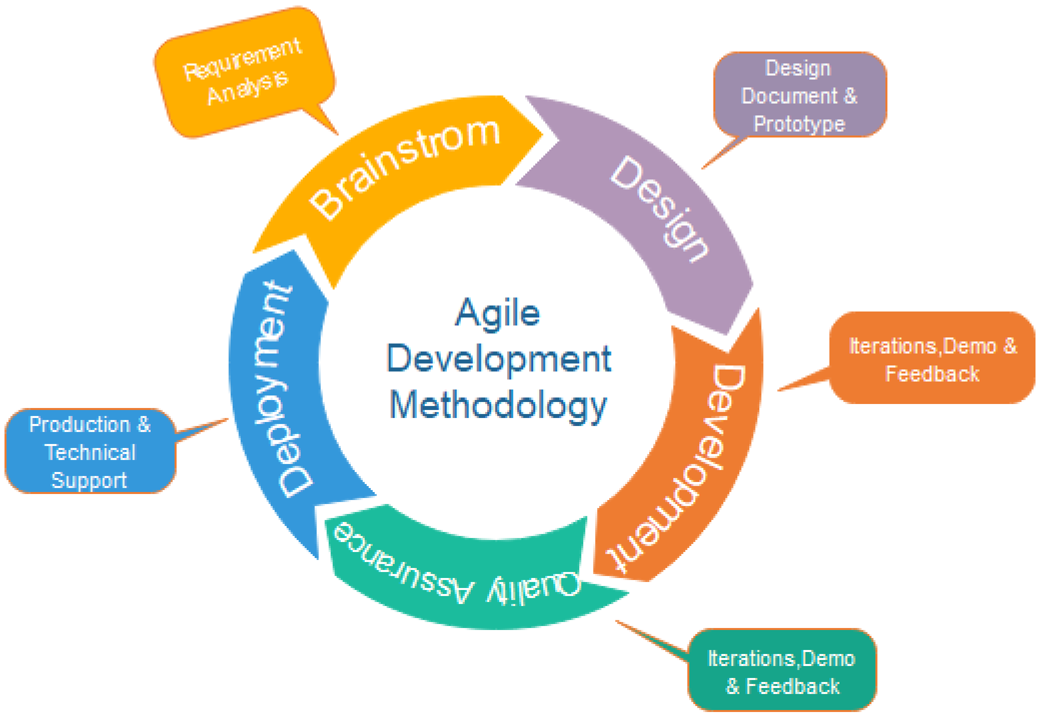

Requirement Analysis
Brainstrom
Design Document & Prototype
Design
Iterations,Demo & Feedback
Development
Agile Development Methodology
Production & Technical Support
Deployment
Quality Assurance
Iterations,Demo & Feedback

Fig. Agile Model

2. Design the requirements: When you have identified the project, work with stakeholders to define requirements. You can use the user flow diagram or the high-level UML diagram to show the work of new features and show how it will apply to your existing system.

3. Construction/ iteration: When the team defines the requirements, the work begins. Designers and developers start working on their project, which aims to deploy a working product. The product will undergo various stages of improvement, so it includes simple, minimal functionality.

4. Testing: In this phase, the Quality Assurance team examines the product's performance and looks for the bug.

5. Deployment: In this phase, the team issues a product for the user's work environment.

6. Feedback: After releasing the product, the last step is feedback. In this, the team receives feedback about the product and works through the feedback.

Agile Testing Methods:

1. Scrum

2. Crystal

3. Dynamic Software Development Method(DSDM)

4. Feature Driven Development(FDD)

5. Lean Software Development

6.Atern

Scrum

SCRUM is an agile development process focused primarily on ways to manage tasks in team-based development conditions.

There are three roles in it, and their responsibilities are:

Scrum Master: The scrum can set up the master team, arrange the meeting and remove obstacles for the process

Product owner: The product owner makes the product backlog, prioritizes the delay and is responsible for the distribution of functionality on each repetition.

Scrum Team: The team manages its work and organizes the work to complete the sprint or cycle.

Extreme programming (XP):

It uses specific practices like pair programming, continuous integration, and test-driven development to achieve these goals. Extreme programming is ideal for projects that have high levels of uncertainty and require frequent changes, as it allows for quick adaptation to new requirements and feedback.

MODULE-3

CODING AND SOFTWARE TESTING TECHNIQUES

Coding

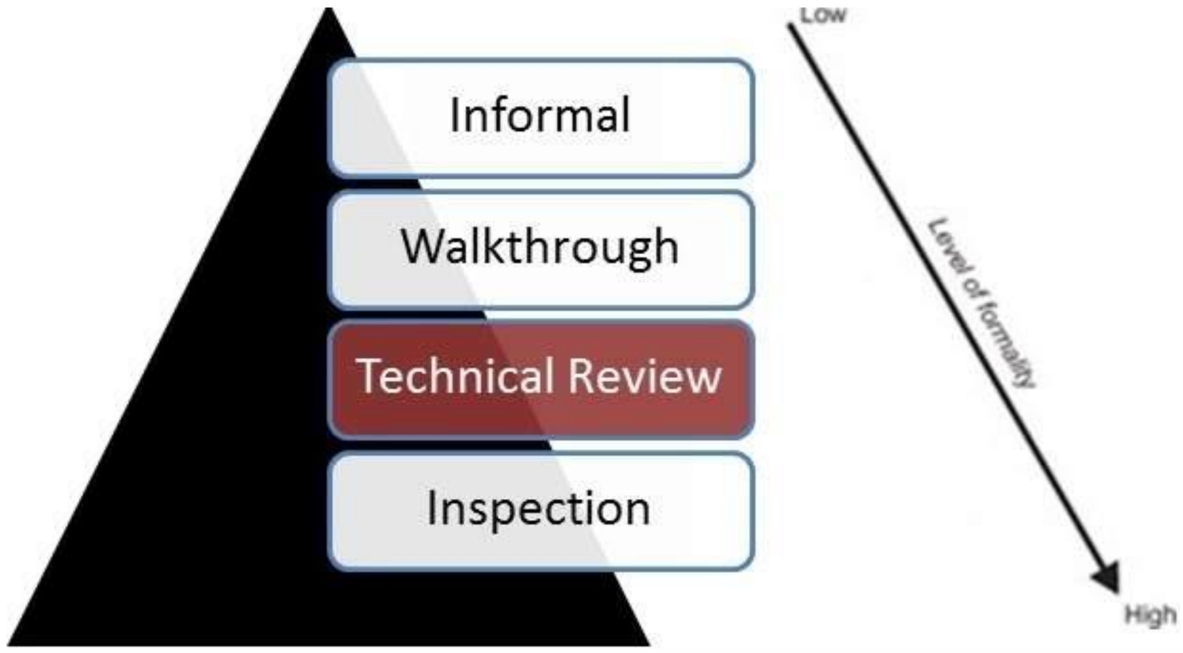
Coding, sometimes called computer programming, is how we communicate with computers.

Code tells a computer what actions to take, and writing code is like creating a set of instructions.

Code Review

Code Review is a systematic examination, which can find and remove the vulnerabilities in the code such as memory leaks and buffer overflows**.**

Where Code Review fits in ?



Why are code reviews important?

Developing a strong code review process sets a foundation for continuous improvement and prevents unstable code from shipping to customers. Code reviews should become part of a software development team's workflow to improve code quality and ensure that every piece of code has been looked at by another team member.

What are the benefits of code reviews?

1. Share knowledge: When software developers review code as soon as a team member makes changes, they can learn new techniques and solutions. Code reviews help junior developers learn from more senior team members, similar to how pair programming effectively helps developers share skills and ideas. By spreading knowledge across the organization, code reviews ensure that no person is a single point of failure. Everyone has the ability to review and offer feedback. Shared knowledge also helps team members take vacation, because everyone on the team has background knowledge on a topic.

2. Discover bugs earlier: Rather than discovering bugs after a feature has been shipped and scrambling to release a patch, developers can immediately find and fix problems before customers ever see them. Moving the review process earlier in the software development lifecycle through unit tests helps developers work on fixes with fresh knowledge. When waiting until the end of the lifecycle to do a review, developers often struggle to remember code, solutions, and reasoning. Static analysis is a cheap, efficient way to meet business and customer value.

3 Maintain compliance: Developers have various backgrounds and training that influence their coding styles. If teams want to have a standard coding style, code reviews help everyone adhere to the same standards. This is especially important for open source projects that have multiple individuals contributing code. Peer reviews bring in maintainers to assess the code before pushing changes.

4. Enhance security: Code reviews create a high level of security, especially when security professionals engage in a targeted review. Application security is integral in software development, and code reviews help detect security issues and ensure compliance. Security team members can review code for vulnerabilities and alert developers to the threat. Code reviews are a great complement to automated scans and tests that detect security vulnerabilities.

5. Increase collaboration: When team members work together to create a solution, they feel more ownership of their work and a stronger sense of belonging. Authors and reviewers can work together to find the most effective solutions to meet customer needs. It’s important to strengthen collaboration across the software development lifecycle to prevent information silos and maintain a seamless workflow between teams. To successfully conduct code reviews, it’s important that developers build a code review mindset that has a strong foundation in collaborative development.

6. Improve code quality: Code reviews are an important way to ensure you ship high-quality code and quality software. A human who knows your code base can notice code quality issues that automated tests may miss. They can even help you reduce technical debt.

What are the disadvantages of code reviews?

1. Longer time to ship: The review time could delay the release process, since reviewers have to collaborate with authors to discuss problems. Depending on a reviewer's workload, they may not complete a review as fast as the author would like. This challenge can be overcome by using code review tools that include automated testing to find errors. Automated tooling is an effective way to free up developer time so that they can focus on the larger software engineering problems rather than highlight simple lint errors.

2. Pull focus from other tasks: Developers often have a heavy workload, and a code review can pull their focus away from other high priority tasks that they're responsible for delivering. Team members may be forced to decide between completing their task or halting their work in order to do a code review. In either case, work is delayed somewhere across the organization. To reduce this pain point, team members can have a reviewer roulette or a list of domain experts so that a single developer isn't inundated with review requests.

3. Large reviews mean longer review times: If developers have to conduct code reviews on a large change, they could spend a significant amount of time examining the code. Large code reviews are challenging to assess, and developers may naturally move through the process quickly in order to complete it in a timely manner, resulting in decreased feedback quality. Incremental code development prevents this challenge by enabling reviewers to look at a small piece of code several times rather than a large change at once.

Four approaches to code review

Some of these disadvantages can be minimized by selecting the most appropriate code review method for your team. Here are four common approaches to code review:

Pair programming

Pair programming involves two developers collaborating in real time — one writing code (the driver) and one reviewing code (the navigator). Pairing sessions are popular with development teams because teammates collaborate to identify the most effective solution to a challenge. Team members share knowledge and can quickly overcome difficulties by working through ideas together and drawing on their expertise.

The benefits of pair programming

1. Transfers knowledge

2. Prevents information silos

3. Solves complex problems

4. Increases morale

5. Finds more bugs

6. Can be conducted remotely

The drawbacks of pair programming

1. Time-consuming

2. Can be overused

3. Difficult to measure

Over-the-shoulder reviews

In an over-the-shoulder-review, two developers — the author and reviewer — team up in person or remotely through a shared screen and the author explains the completed change proposal and offers reasoning for the chosen solutions. The reviewer asks questions and makes suggestions, similar to how team members collaborate during pairing sessions. The author can make small changes during the review and note larger fixes for a later time.

The benefits of over-the-shoulder reviews

Easy implementation and completion

Can be conducted remotely

Faster than pair programming

The drawbacks of over-the-shoulder reviews

Reviewer is detached from code

Review moves at the author's pace

Lack of objectivity

No verification that changes were made

Difficult to measure

Tool-assisted reviews

Teams may decide to use tools to save time and ensure the highest quality code is shipped. Tool-assisted reviews can automatically gather changed files and display the differences, or make it easier to provide feedback and have conversations via comments, and incorporate things like static application security testing (SAST) to help identify and remediate vulnerabilities.

The best way to look at tool-assisted reviews is to consider them a complement to other types of reviews. Automated tooling is an effective way to enforce code standards, identify vulnerability, gather metrics, and gather files, but some teams may be tempted to completely rely on tooling and forgo team member involvement to conduct code reviews. Tools should be viewed as an extension of code reviews and a way to enhance the process.

The benefits of tool-assisted reviews

Easier to gather metrics

Automated tooling frees up developer focus

The drawbacks of tool-assisted reviews

Developers must maintain tools

Expensive

Will still require teammate reviews

Email pass-around

Email pass-arounds are often used for minor difficulties and small pieces of code. They can be conducted via email or source code management systems. During an email pass-around, an author sends an email containing code changes to reviewers. Email pass-around is similar to over-the-shoulder reviews in that they can be easily implemented and don't require a strong learning curve or a mentoring stage to teach the author how to make a change.

The benefits of email pass-arounds

Easy implementation and completion

Facilitates remote, asynchronous reviews

Automatic reviews via SCMs

The drawbacks of email pass-arounds

Time consuming to gather files

Difficult to follow conversations

No definite review end date

No verification that changes were made

Software Documentation

Software documentation is a written piece of text that is often accompanied by a software program.

Types Of Software Documentation :

1. Requirement Documentation: It is the description of how the software shall perform and which environment setup would be appropriate to have the best out of it. These are generated while the software is under development and is supplied to the tester groups too.

2. Architectural Documentation: Architecture documentation is a special type of documentation that concerns the design. It contains very little code and is more focused on the components of the system, their roles, and working. It also shows the data flow throughout the system**.**

3. Technical Documentation: These contain the technical aspects of the software like API, algorithms, etc. It is prepared mostly for software devs.

4. End-user Documentation: As the name suggests these are made for the end user. It contains support resources for the end user.

Importance of software documentation :

For a programmer reliable documentation is always a must the presence keeps track of all aspects of an application and helps in keeping the software updated.

Advantages of software documentation :

1. The presence of documentation helps in keeping the track of all aspects of an application and also improves the quality of the software product.

2. The main focus is based on the development, maintenance, and knowledge transfer to other developers.

3. Helps development teams during development.

4. Helps end-users in using the product.

5. Improves overall quality of software product

6. It cuts down duplicative work.

7. Makes easier to understand code.

8. Helps in establishing internal coordination in work.

Disadvantages of software documentation :

1.The documenting code is time-consuming.

2. The software development process often takes place under time pressure, due to which many times the documentation updates don't match the updated code.

3. The documentation has no influence on the performance of an application.

4. Documenting is not so fun, it's sometimes boring to a certain extent.

Testing

Unit Testing

Unit testing is a type of software testing that focuses on individual units or components of a software system.

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.

The purpose of unit testing is to validate that each unit of the software works as intended and meets the requirements.

It is correlated with the functional correctness of the independent modules.

Unit testing is such a type of testing technique that is usually performed by developers.

Objective of Unit Testing:

The objective of Unit Testing is:

1. To isolate a section of code.

2. To verify the correctness of the code.

3. To test every function and procedure.

4. To fix bugs early in the 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 with code reuse.

Types of Unit Testing:

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

Workflow of Unit Testing: Unit Testing Techniques:

There are 3 types of Unit Testing Techniques. They are

Black Box Testing: This testing technique is used in covering the unit tests for input, user interface, and output parts.

White Box Testing: This technique is used in testing the functional behavior of the system by giving the input and checking the functionality output including the internal design structure and code of the modules.

Gray Box Testing: This technique is used in executing the relevant test cases, test methods, test functions, and analyzing the code performance for the modules.

Unit Testing Tools:

Here are some commonly used Unit Testing tools:

Jtest

Junit

NUnit

EMMA

PHPUnit

Advantages of Unit Testing:

1. 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.

2. Unit testing allows the programmer to refine code and make sure the module works properly.

3. Unit testing enables testing parts of the project without waiting for others to be completed.

4. Early Detection of Issues: Unit testing allows developers to detect and fix issues early in the development process, before they become larger and more difficult to fix.

5. Improved Code Quality: Unit testing helps to ensure that each unit of code works as intended and meets the requirements, improving the overall quality of the software.

6. Increased Confidence: Unit testing provides developers with confidence in their code, as they can validate that each unit of the software is functioning as expected.

7. Faster Development: Unit testing enables developers to work faster and more efficiently, as they can validate changes to the code without having to wait for the full system to be tested.

8. Better Documentation: Unit testing provides clear and concise documentation of the code and its behavior, making it easier for other developers to understand and maintain the software.

9. Facilitation of Refactoring: Unit testing enables developers to safely make changes to the code, as they can validate that their changes do not break existing functionality.

10. Reduced Time and Cost: Unit testing can reduce the time and cost required for later testing, as it helps to identify and fix issues early in the development process.

Disadvantages of Unit Testing:

1. The process is time-consuming for writing the unit test cases.

2. Unit Testing will not cover all the errors in the module because there is a chance of having errors in the modules while doing integration testing.

3. Unit Testing is not efficient for checking the errors in the UI(User Interface) part of the module.

4. It requires more time for maintenance when the source code is changed frequently.

5. It cannot cover the non-functional testing parameters such as scalability, the performance of the system, etc.

6. Time and Effort: Unit testing requires a significant investment of time and effort to create and maintain the test cases, especially for complex systems.

7. Dependence on Developers: The success of unit testing depends on the developers, who must write clear, concise, and comprehensive test cases to validate the code.

8. Difficulty in Testing Complex Units: Unit testing can be challenging when dealing with complex units, as it can be difficult to isolate and test individual units in isolation from the rest of the system.

9. Difficulty in Testing Interactions: Unit testing may not be sufficient for testing interactions between units, as it only focuses on individual units.

10. Difficulty in Testing User Interfaces: Unit testing may not be suitable for testing user interfaces, as it typically focuses on the functionality of individual units.

11. Over-reliance on Automation: Over-reliance on automated unit tests can lead to a false sense of security, as automated tests may not uncover all possible issues or bugs.

12. Maintenance Overhead: Unit testing requires ongoing maintenance and updates, as the code and test cases must be kept up-to-date with changes to the software.

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 the 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 a 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 types of inputs work similarly so instead of giving all of them separately we can group them and test only one input of each group. The idea is to partition the input domain of the system into several equivalence classes such that each member of the class works similarly, i.e., if a test case in one class results in some error, other members of the class would also result in the same error.

The technique involves two steps:

1. Identification of equivalence class – Partition any input domain into a minimum of 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 a unique identification number. (ii) Write a test case covering all valid and invalid test cases 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:

The whole number which is a perfect square- output will be an integer.

The whole number which is not a perfect square- output will be a decimal number.

Positive decimals

Negative numbers(integer or decimal).

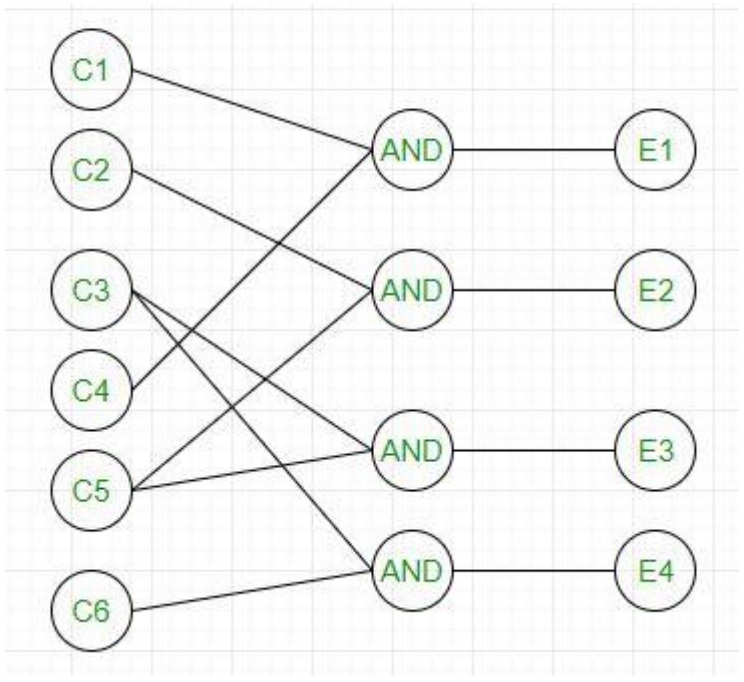
Characters other than numbers like “a”,”!”,”;”, etc.

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

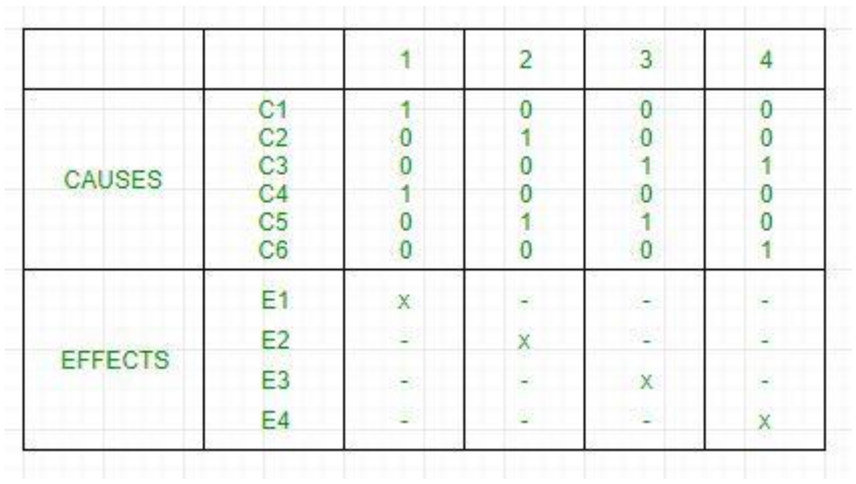
4. Cause effect Graphing – This technique establishes a relationship between logical input called causes with corresponding actions called the effect. The causes and effects are represented using Boolean graphs. The following steps are followed:

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

For example, in the following cause-effect graph:



It can be converted into a 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 the SRS of a software system.

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

Processor (Pentium 3, Pentium 4) and several processors.

Architecture and characteristics of machine (32-bit or 64-bit).

Back-end components such as database servers.

Operating System (Windows, Linux, etc).

Black Box Testing Type

The following are the several categories of black box testing:

1. Functional Testing

2. Regression Testing

3. Nonfunctional Testing (NFT)

Functional Testing: It determines the system's software functional requirements.

Regression Testing: It ensures that the newly added code is compatible with the existing code. In other words, a new software update has no impact on the functionality of the software. This is carried out after a system maintenance operation and upgrades.

Nonfunctional Testing: Nonfunctional testing is also known as NFT. This testing is not functional testing of software. It focuses on the software's performance, usability, and scalability.

Tools Used for Black Box Testing:

Appium

Selenium

Microsoft Coded UI

Applitools

HP QTP.

Advantages of Black Box Testing:

1. The tester does not need to have more functional knowledge or programming skills to implement the Black Box Testing.

2. It is efficient for implementing the tests in the larger system.

3. Tests are executed from the user's or client's point of view.

4. Test cases are easily reproducible.

5. It is used in finding the ambiguity and contradictions in the functional specifications.

Disadvantages of Black Box Testing:

1. There is a possibility of repeating the same tests while implementing the testing process.

2. Without clear functional specifications, test cases are difficult to implement.

3. It is difficult to execute the test cases because of complex inputs at different stages of testing.

4. Sometimes, the reason for the test failure cannot be detected.

5. Some programs in the application are not tested.

6. It does not reveal the errors in the control structure.

7. Working with a large sample space of inputs can be exhaustive and consumes a lot of time.

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.

White Box Testing is also known as transparent testing, open box testing.

White box testing is a software testing technique that involves testing the internal structure and workings of a software application.

The tester has access to the source code and uses this knowledge to design test cases that can verify the correctness of the software at the code level.

White box testing isused to test the software's internal logic, flow, and structure.

Working process of white box testing:

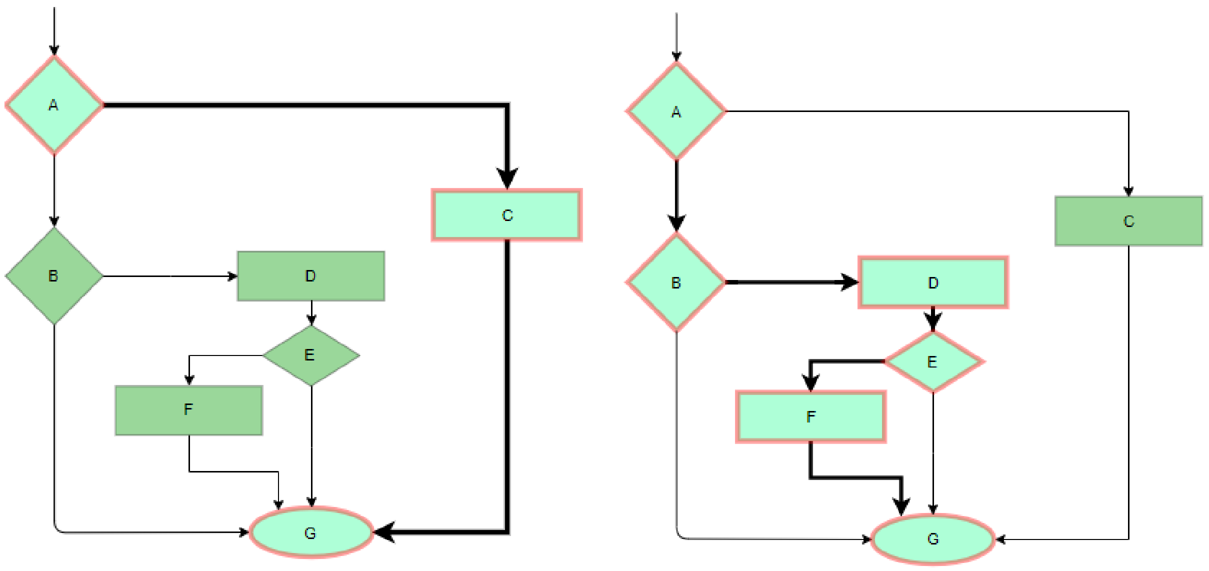
1.Input: Requirements, Functional specifications, design documents, source code.

2. Processing: Performing risk analysis for guiding through the entire process.

3. Proper test planning: Designing test cases so as to cover the entire code. Execute rinse-repeat until error-free software is reached. Also, the results are communicated.

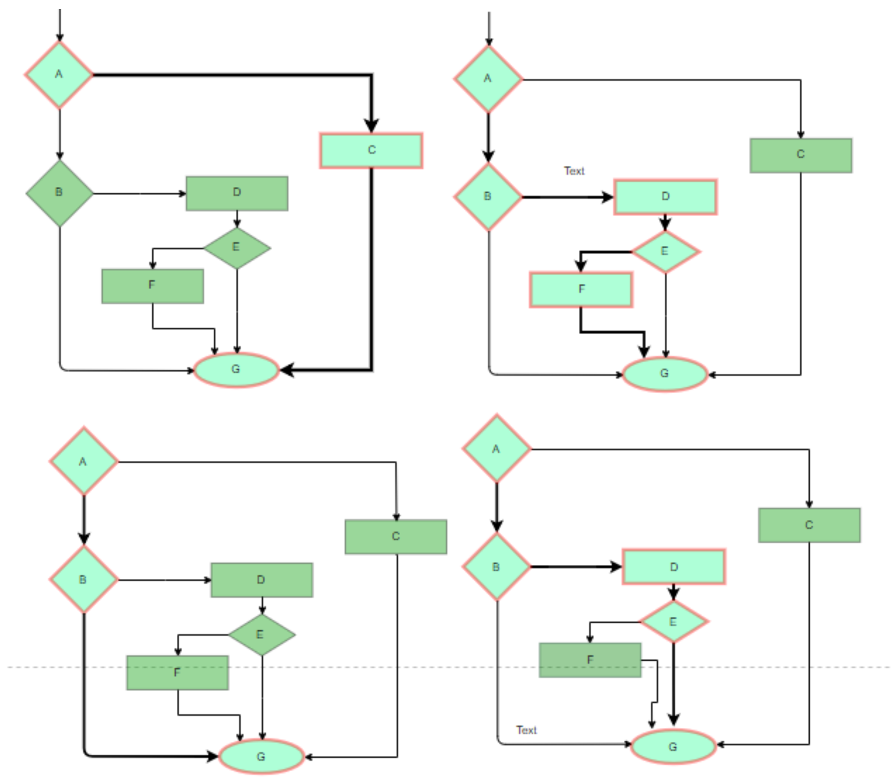
4. Output: Preparing final report of the entire testing process.

- **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.



Statement Coverage Example

- **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.



4 test cases required such that all branches of all decisions are covered, i.e, all edges of flowchart are covered

- **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’
    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’
    4. #TC1: X = 0, Y = 0
    5. #TC2: X = 0, Y = 5
    6. #TC3: X = 55, Y = 0
    7. #TC4: X = 55, Y = 5
- **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
    5. $V(G) = P + 1$, where P is the number of predicate nodes in the flow graph
    6. $V(G) = E – N + 2$, where E is the number of edges and N is the total number of nodes

7. V(G) = Number of non-overlapping regions in the graph
8. #P1: 1 – 2 – 4 – 7 – 8
9. #P2: 1 – 2 – 3 – 5 – 7 – 8
10. #P3: 1 – 2 – 3 – 6 – 7 – 8
11. #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

Tools required for White box Testing:

PyUnit

Sqlmap

Nmap

Parasoft Jtest

Nunit

VeraUnit

CppUnit

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.

5. White box testing can be easily started in Software Development Life Cycle.

6. Easy Code Optimization.

Disadvantages:

1. Redesign of code and rewriting code needs test cases to be written again.

2. Testers are required to have in-depth knowledge of the code and programming language as opposed to black box testing.

3. Missing functionalities cannot be detected as the code that exists is tested.

4. Very complex and at times not realistic.

5. Much more chances of Errors in production.

6. It is very expensive.

Cyclomatic Complexity

Cyclomatic complexity of a code section is the quantitative measure of the number of linearly independent paths in it.

It is a software metric used to indicate the complexity of a program.

It is computed using the Control Flow Graph of the program. The nodes in the graph indicate the smallest group of commands of a program, and a directed edge in it connects the two nodes i.e. if second command might immediately follow the first command.

For example, if source code contains no control flow statement then its cyclomatic complexity will be 1 and source code contains a single path in it. Similarly, if the source code contains one if condition then cyclomatic complexity will be 2 because there will be two paths one for true and the other for false.

Mathematically, for a structured program, the directed graph inside control flow is the edge joining two basic blocks of the program as control may pass from first to second.

So, cyclomatic complexity M would be defined as,

M = E – N + 2P

where,

E = the number of edges in the control flow graph

N = the number of nodes in the control flow graph

P = the number of connected components

Steps that should be followed in calculating cyclomatic complexity and test cases design are:

1. Construction of graph with nodes and edges from code.

2. Identification of independent paths.

3. Cyclomatic Complexity Calculation

4. Design of Test Cases

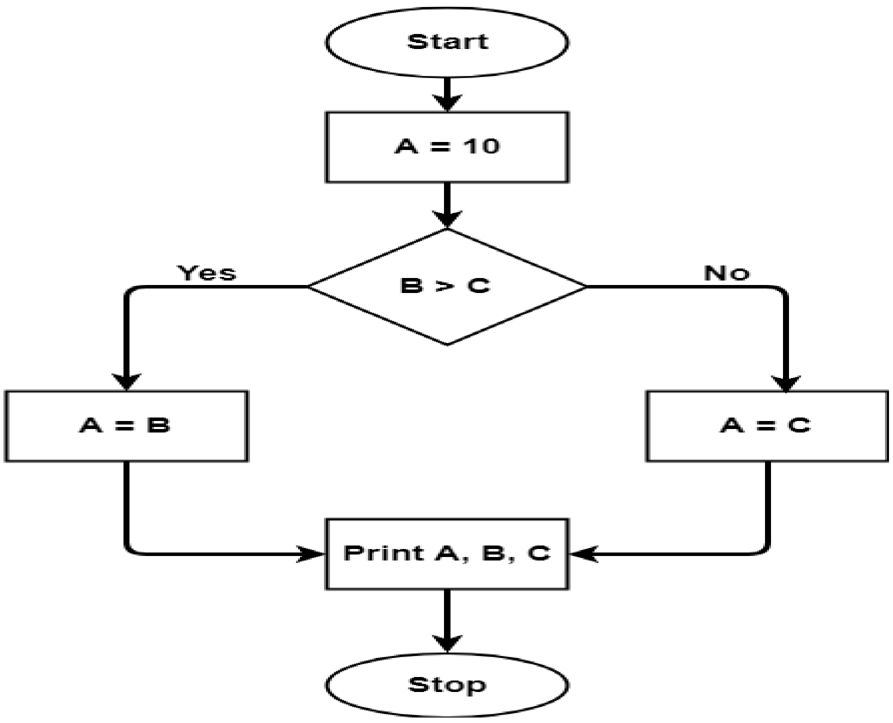
Let a section of code as such:

```
A = 10
  IF B > C THEN
    A = B
  ELSE
    A = C
  ENDIF
Print A
```

Print B

Print C

Control Flow Graph of above code



cyclomatic-complexity

The cyclomatic complexity calculated for above code will be from control flow graph. The graph shows seven shapes(nodes), seven lines(edges), hence cyclomatic complexity is 7-7+2 = 2.

Use of Cyclomatic Complexity:

Determining the independent path executions thus proven to be very helpful for Developers and Testers.

It can make sure that every path have been tested at least once.

Thus help to focus more on uncovered paths.

Code coverage can be improved.

Risk associated with program can be evaluated.

These metrics being used earlier in the program helps in reducing the risks.

Advantages of Cyclomatic Complexity:.

1. It can be used as a quality metric, gives relative complexity of various designs.

2. It is able to compute faster than the Halstead's metrics.

3. It is used to measure the minimum effort and best areas of concentration for testing.

4. It is able to guide the testing process.

5. It is easy to apply.

Disadvantages of Cyclomatic Complexity:

1. It is the measure of the programs's control complexity and not the data complexity.

2. In this, nested conditional structures are harder to understand than non-nested structures.

3. In case of simple comparisons and decision structures, it may give a misleading figure.

Coverage Analysis

Code coverage is a software testing metric or also termed as a Code Coverage Testing which helps in determining how much code of the source is tested which helps in accessing quality of test suite and analyzing how comprehensively a software is verified.

Actually in simple code coverage refers to the degree of which the source code of the software code has been tested. This Code Coverage is considered as one of the form of white box testing.

Code Coverage metric helps in determining the performance and quality aspects of any software.

The formula to calculate code coverage is

Code Coverage = (Number of lines of code executed)/(Total Number of lines of code in a system component) * 100

Code Coverage Criteria :

To perform code coverage analysis various criteria are taken into consideration. These are the major methods/criteria which are considered.

1. Statement Coverage/Block coverage :

The number of statements that have been successfully executed in the program source code.

Statement Coverage = (Number of statements executed)/(Total Number of statements)*100.

2. Decision Coverage/Branch Coverage :

The number of decision control structures that have been successfully executed in the program source code.

Decision Coverage = (Number of decision/branch outcomes exercised)/(Total number of decision outcomes in the source code)*100.

3. Function coverage :

The number of functions that are called and executed at least once in the source code.

Function Coverage = (Number of functions called)/(Total number of function)*100

4. Condition Coverage/Expression Coverage :

The number of Boolean condition/expression statements executed in the conditional statement.

Condition Coverage =(Number of executed operands)/(Total Number of Operands)*100.

Tools For Code Coverage :

Below are the few important code coverage tools

Cobertura

Clover

Gretel

Kalistick

JaCoCo

JTest

OpenCover

Emma

GCT

Advantages of Using Code Coverage :

1. It helps in determining the performance and quality aspects of any software.

2. It helps in evaluating quantitative measure of code coverage.

3. It helps in easy maintenance of code base.

4. It helps in accessing quality of test suite and analyzing how comprehensively a software is verified.

5. It helps in exposure of bad, dead, and unused code.

6. It helps in creating extra test cases to increase coverage.

7. It helps in developing the software product faster by increasing its productivity and efficiency.

8. It helps in measuring the efficiency of test implementation.

9. It helps in finding new test cases which are uncovered.

Disadvantages of Using Code Coverage :

1. Some times it fails to cover code completely and correctly.

2. It can not guarantee that all possible values of a feature is tested with the help of code coverage.

3. It fails in ensuring how perfectly the code has been covered.

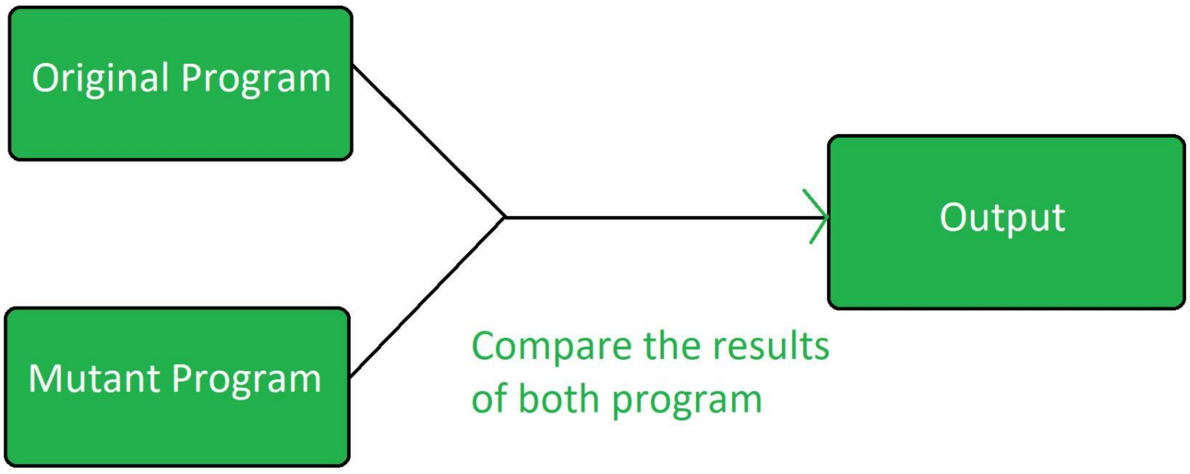
Mutation Testing

Mutation Testing is a type of Software Testing that is performed to design new software tests and also evaluate the quality of already existing software tests.

Mutation testing is related to modification a program in small ways.

It focuses to help the tester develop effective tests or locate weaknesses in the test data used for the program.

Mutation Testing is a White Box Testing.



Mutation testing can be applied to design models, specifications, databases, tests, and XML. It is a structural testing technique, which uses the structure of the code to guide the testing process. It can be described as the process of rewriting the source code in small ways in order to remove the redundancies in the source code.

What is mutation?

The mutation is a small modification in a program; these minor modifications are planned to typical low-level errors which are happened at the time of coding process.

History of Mutation Testing:

Richard Lipton proposed the mutation testing in 1971 for the first time. Although high cost reduced the use of mutation testing but now it is widely used for languages such as Java and XML.

Objective of Mutation Testing:

1. The objective of mutation testing is:

2. To identify pieces of code that are not tested properly.

3. To identify hidden defects that can't be detected using other testing methods.

4. To discover new kinds of errors or bugs.

5. To calculate the mutation score.

6. To study error propagation and state infection in the program.

7. To assess the quality of the test cases.

Types of Mutation Testing:

Mutation testing is basically of 3 types:

1. Value Mutations:

In this type of testing the values are changed to detect errors in the program. Basically a small value is changed to a larger value or a larger value is changed to a smaller value. In this testing basically constants are changed.

Example:

Initial Code:

```
int mod = 1000000007;

int a = 12345678;

int b = 98765432;

int c = (a + b) % mod;
```

Changed Code:

```
int mod = 1007;

int a = 12345678;

int b = 98765432;

int c = (a + b) % mod;
```

2. Decision Mutations:

In decisions mutations are logical or arithmetic operators are changed to detect errors in the program.

Example:

Initial Code:

```
if(a < b)
 c = 10;
else
 c = 20;
```

Changed Code:

```
if(a > b)
 c = 10;
else
 c = 20;
```

3. Statement Mutations:

In statement mutations a statement is deleted or it is replaces by some other statement.

Example:

Initial Code:

```
if(a < b)
 c = 10;
else
 c = 20;
```

Changed Code:

```
if(a < b)
 d = 10;
```

else

d = 20;

Tools used for Mutation Testing :

1. Judy

2. Jester

3. Jumble

4. PIT

5. MuClipse.

Advantages of Mutation Testing:

1. It brings a good level of error detection in the program.

2, It discovers ambiguities in the source code.

3. It finds and solves the issues of loopholes in the program.

4. It helps the testers to write or automate the better test cases.

5. It provides more efficient programming source code.

Disadvantages of Mutation Testing:

1. It is highly costly and time-consuming.

2. It is not able for Black Box Testing.

3. Some, mutations are complex and hence it is difficult to implement or run against various test cases.

4. Here, the team members who are performing the tests should have good programming knowledge.

5. Selection of correct automation tool is important to test the programs.

Debugging :

Debugging is the process of finding and resolving defects or problems within a computer program that prevent correct operation of computer software or a system.

Need for debugging:

Once errors are known during a program code, it’s necessary to initial establish the precise program statements liable for the errors and so to repair them.

Challenges in Debugging:

There are lot of problems at the same time as acting the debugging. These are the following:

1. Debugging is finished through the individual that evolved the software program and it’s miles difficult for that person to acknowledge that an error was made.

2. Debugging is typically performed under a tremendous amount of pressure to fix the supported error as quick as possible.

3. It can be difficult to accurately reproduce input conditions.

4. Compared to the alternative software program improvement activities, relatively little research, literature and formal preparation exist at the procedure of debugging.

Debugging Approaches:

The following are a number of approaches popularly adopted by programmers for debugging**.**

1. Brute Force Method:

This is the foremost common technique of debugging however is that the least economical method. during this approach, the program is loaded with print statements to print the intermediate values with the hope that a number of the written values can facilitate to spot the statement in error. This approach becomes a lot of systematic with the utilisation of a symbolic program (also known as a source code debugger), as a result of values of various variables will be simply checked and breakpoints and watch-points can be easily set to check the values of variables effortlessly.

2.**Backtracking:**

This is additionally a reasonably common approach. during this approach, starting from the statement at which an error symptom has been discovered, the source code is derived backward till the error is discovered. sadly, because the variety of supply lines to be derived back will increase, the quantity of potential backward methods will increase and should become unimaginably large so limiting the utilisation of this approach.

3. Cause Elimination Method:

In this approach, a listing of causes that may presumably have contributed to the error symptom is developed and tests are conducted to eliminate every error. A connected technique of identification of the error from the error symptom is that the package fault tree analysis.

4. Program Slicing:

This technique is analogous to backtracking. Here the search house is reduced by process slices. A slice of a program for a specific variable at a particular statement is that the set of supply lines preceding this statement which will influence the worth of that variable

Integration Testing

Integration testing is a software testing technique that focuses on verifying the interactions and data exchange between different components or modules of a software application.

The goal of integration testing is to identify any problems or bugs that arise when different components are combined and interact with each other.

Integration testing is typically performed after unit testing and before system testing. It helps to identify and resolve integration issues early in the development cycle, reducing the risk of more severe and costly problems later on.

Exposing the defects is the major focus of the integration testing and the time of interaction between the integrated units.

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 combined and the functionality is verified 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 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 is very expensive to fix.

Big-Bang integration testing is a software testing approach in which all components or modules of a software application are combined and tested at once.

This approach is typically used when the software components have a low degree of interdependence or when there are constraints in the development environment that prevent testing individual components.

The goal of big-bang integration testing is to verify the overall functionality of the system and to identify any integration problems that arise when the components are combined.

While big-bang integration testing can be useful in some situations, it can also be a high-risk approach, as the complexity of the system and the number of interactions between components can make it difficult to identify and diagnose problems.

Advantages:

1. It is convenient for small systems.
2. Simple and straightforward approach.
3. Can be completed quickly.
4. Does not require a lot of planning or coordination.
5. May be suitable for small systems or projects with a low degree of interdependence between components.

Disadvantages:

1. There will be quite a lot of delay because you would have to wait for all the modules to be integrated.
2. High risk critical modules are not isolated and tested on priority since all modules are tested at once.
3. Not Good for long Projects.
4. High risk of integration problems that are difficult to identify and diagnose.
5. Can result in long and complex debugging and troubleshooting efforts.

6. Can lead to system downtime and increased development costs.

7. May not provide enough visibility into the interactions and data exchange between components.

8. Can result in a lack of confidence in the system's stability and reliability.

9. Can lead to decreased efficiency and productivity.

10. May result in a lack of confidence in the development team.

11. Can lead to system failure and decreased user satisfaction.

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 that each subsystem tests 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:

1. In bottom-up testing, no stubs are required.

2. A principle advantage of this integration testing is that several disjoint subsystems can be tested simultaneously.

3. It is easy to create the test conditions.

4. Best for the applications that uses bottom up design approach.

5. It is Easy to observe the test results.

Disadvantages:

1. Driver modules must be produced.

2. In this testing, the complexity that occurs when the system is made up of a large number of small subsystems.

3. As Far modules have been created, there is no working model can be represented.

3. Top-Down Integration Testing – Top-down integration testing technique is 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:

1. Separately debugged module.

2. Few or no drivers needed.

3. It is more stable and accurate at the aggregate level.

4. Easier isolation of interface errors.

5. In this, design defects can be found in the early stages.

Disadvantages:

1. Needs many Stubs.

2. Modules at lower level are tested inadequately.

3. It is difficult to observe the test output.

4. It is difficult to stub design.

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. It is also called the hybrid integration testing. also, stubs and drivers are used in mixed integration testing.

Advantages:

1. Mixed approach is useful for very large projects having several sub projects.

2. This Sandwich approach overcomes this shortcoming of the top-down and bottom-up approaches.

3. Parallel test can be performed in top and bottom layer tests.

Disadvantages:

1. For mixed integration testing, it requires very high cost because one part has Top-down approach while another part has bottom-up approach.

2. This integration testing cannot be used for smaller systems with huge interdependence between different modules.

System Testing

System testing is a type of software testing that evaluates the overall functionality and performance of a complete and fully integrated software solution.

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.

The goal of integration testing is to detect any irregularity between the units that are integrated togetherthe corresponding requirements.

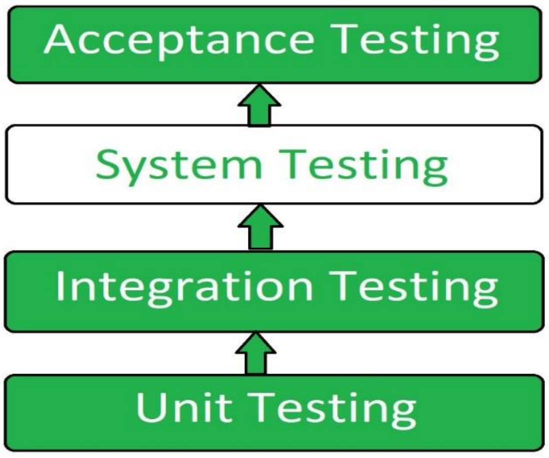
The result of system testing is the observed behavior of a component or a system when it is tested.

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 behavior 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.

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.

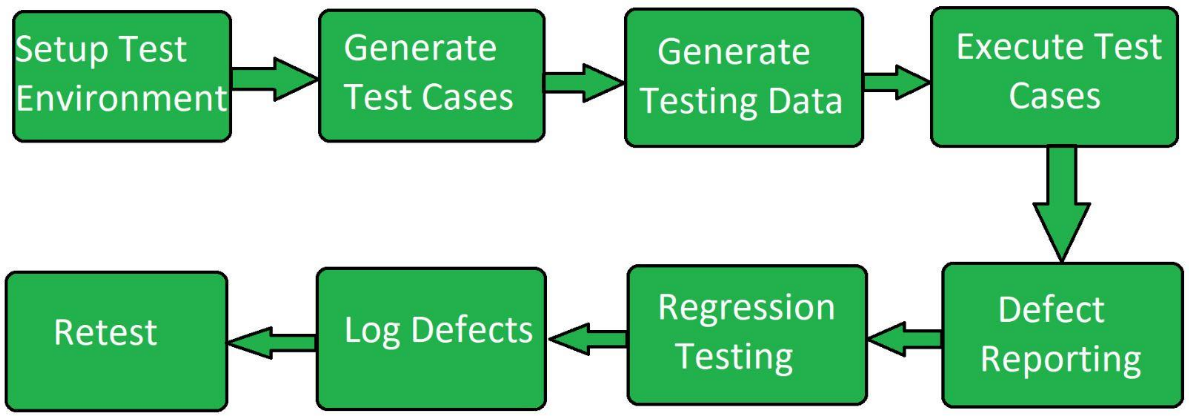
Execute Test Case: After the generation of the test case and the test data, test cases are executed.

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 behavior 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.

Tools used for System Testing :

1. JMeter

2. Gallen Framework

3. Selenium

Here are a few common tools used for System Testing:

1. HP Quality Center/ALM

2. IBM Rational Quality Manager

3. Microsoft Test Manager

4. Selenium

5. Appium

6. LoadRunner

7. Gatling

8. JMeter

9. Apache JServ

10. SoapUI

Here are some advantages of System Testing:

1. Verifies the overall functionality of the system.

2. Detects and identifies system-level problems early in the development cycle.

3. Helps to validate the requirements and ensure the system meets the user needs.

4. Improves system reliability and quality.

5. Facilitates collaboration and communication between development and testing teams.

6. Enhances the overall performance of the system.

7. Increases user confidence and reduces risks.

8. Facilitates early detection and resolution of bugs and defects.

9. Supports the identification of system-level dependencies and inter-module interactions.

10. Improves the system's maintainability and scalability.

Here are some disadvantages of System Testing:

1. Can be time-consuming and expensive.

2. Requires adequate resources and infrastructure.

3. Can be complex and challenging, especially for large and complex systems.

4. Dependent on the quality of requirements and design documents.

5. Limited visibility into the internal workings of the system.

6. Can be impacted by external factors like hardware and network configurations.

7. Requires proper planning, coordination, and execution.

8. Can be impacted by changes made during development.

9. Requires specialized skills and expertise.

10. May require multiple test cycles to achieve desired results.

Regression Testing

Regression Testing is the process of testing the modified parts of the code and the parts that might get affected due to the modifications to ensure that no new errors have been introduced in the software after the modifications have been made.

When to do regression testing?

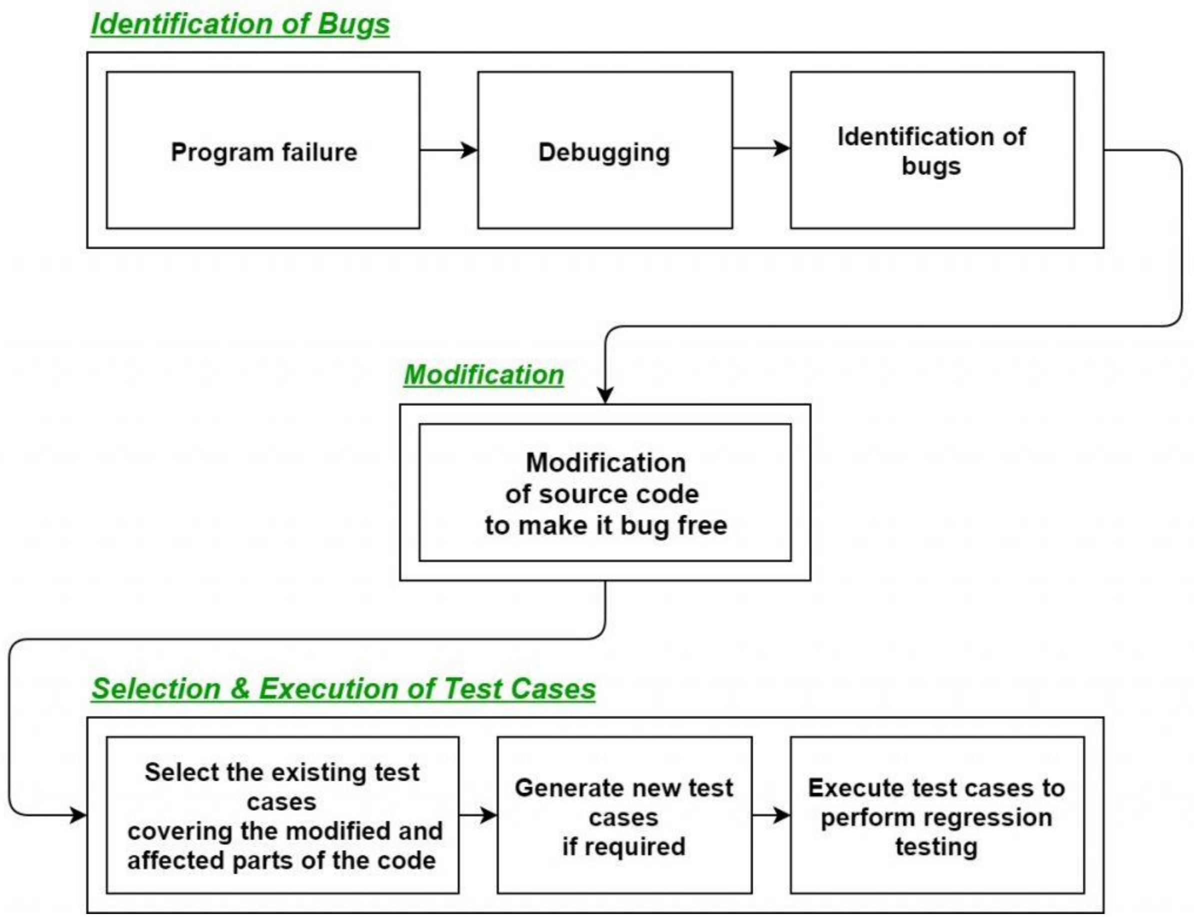
1. When a new functionality is added to the system and the code has been modified to absorb and integrate that functionality with the existing code.

2. When some defect has been identified in the software and the code is debugged to fix it.

3. When the code is modified to optimize its working.

Process of Regression testing:

Firstly, whenever we make some changes to the source code for any reasons like adding new functionality, optimization, etc. then our program when executed fails in the previously designed test suite for obvious reasons. After the failure, the source code is debugged in order to identify the bugs in the program. After identification of the bugs in the source code, appropriate modifications are made. Then appropriate test cases are selected from the already existing test suite which covers all the modified and affected parts of the source code. We can add new test cases if required. In the end regression testing is performed using the selected test cases.


Identification of Bugs
Program failure
Debugging
Identification of bugs
Modification
Modification of source code to make it bug free
Selection & Execution of Test Cases
Select the existing test cases covering the modified and affected parts of the code
Generate new test cases if required
Execute test cases to perform regression testing

Techniques for the selection of Test cases for Regression Testing:

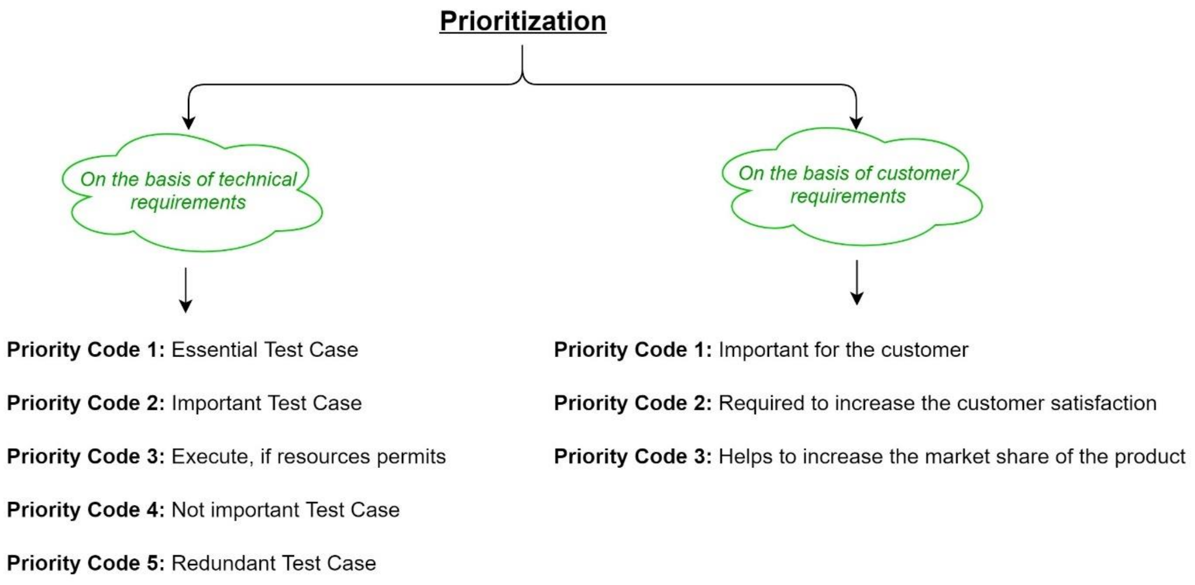
Select all test cases: In this technique, all the test cases are selected from the already existing test suite. It is the most simple and safest technique but not much efficient**.**

Select test cases randomly: In this technique, test cases are selected randomly from the existing test-suite but it is only useful if all the test cases are equally good in their fault detection capability which is very rare. Hence, it is not used in most of the cases.

Select modification traversing test cases: In this technique, only those test cases are selected which covers and tests the modified portions of the source code the parts which are affected by these modifications.

Select higher priority test cases: In this technique, priority codes are assigned to each test case of the test suite based upon their bug detection capability, customer requirements, etc. After assigning the priority codes, test cases with highest priorities are selected for the process of regression testing.

Test case with highest priority has highest rank. For example, test case with priority code 2 is less important than test case with priority code 1.


Prioritization
On the basis of technical requirements
On the basis of customer requirements

Priority Code 1: Essential Test Case

Priority Code 2: Important Test Case

Priority Code 3: Execute, if resources permits

Priority Code 4: Not important Test Case

Priority Code 5: Redundant Test Case

Priority Code 1: Important for the customer

Priority Code 2: Required to increase the customer satisfaction

Priority Code 3: Helps to increase the market share of the product

Tools for regression testing:

Most commonly used tools for regression testing are:

1. Selenium

2. WATIR (Web Application Testing In Ruby)

3. QTP (Quick Test Professional)

4. RFT (Rational Functional Tester)

5. Winrunner

6. Silktest

Advantages of Regression Testing:

1. It ensures that no new bugs has been introduced after adding new functionalities to the system.

2. As most of the test cases used in Regression Testing are selected from the existing test suite and we already know their expected outputs. Hence, it can be easily automated by the automated tools.

3. It helps to maintain the quality of the source code.

Disadvantages of Regression Testing:

1. It can be time and resource consuming if automated tools are not used.

2. It is required even after very small changes in the code.

Software Reliability

1. Software Reliability means Operational reliability. It is described as the ability of a system or component to perform its required functions under static conditions for a specific period.

2. Software reliability is also defined as the probability that a software system fulfills its assigned task in a given environment for a predefined number of input cases, assuming that the hardware and the input are free of error.

3. Software Reliability is an essential connect of software quality, composed with functionality, usability, performance, serviceability, capability, installability, maintainability, and documentation. Software Reliability is hard to achieve because the complexity of software turn to be high. While any system with a high degree of complexity, containing software, will be hard to reach a certain level of reliability, system developers tend to push complexity into the software layer, with the speedy growth of system size and ease of doing so by upgrading the software.

For example, large next-generation aircraft will have over 1 million source lines of software on-board; next-generation air traffic control systems will contain between one and two million lines; the upcoming International Space Station will have over two million lines on-board and over 10 million lines of ground support software; several significant life-critical defense systems will have over 5 million source lines of software. While the complexity of software is inversely associated with software reliability, it is directly related to other vital factors in software quality, especially functionality, capability, etc.

MODULE-4

Maintenance

Basic Concepts in Software Reliability

Software Reliability means Operational reliability. It is described as the ability of a system or component to perform its required functions under static conditions for a specific period.

Software reliability is also defined as the probability that a software system fulfils its assigned task in a given environment for a predefined number of input cases, assuming that the hardware and the input are free of error.

Software Reliability is an essential connect of software quality, composed with functionality, usability, performance, serviceability, capability, installability, maintainability, and documentation.

Software Reliability is hard to achieve because the complexity of software turn to be high.

While any system with a high degree of complexity, containing software, will be hard to reach a certain level of reliability, system developers tend to push complexity into the software layer, with the speedy growth of system size and ease of doing so by upgrading the software.

For example, large next-generation aircraft will have over 1 million source lines of software on-board; next-generation air traffic control systems will contain between one and two million lines; the upcoming International Space Station will have over two million lines on-board and over 10 million lines of ground support software; several significant life-critical defense systems will have over 5 million source lines of software. While the complexity of software is inversely associated with software reliability, it is directly related to other vital factors in software quality, especially functionality, capability, etc.

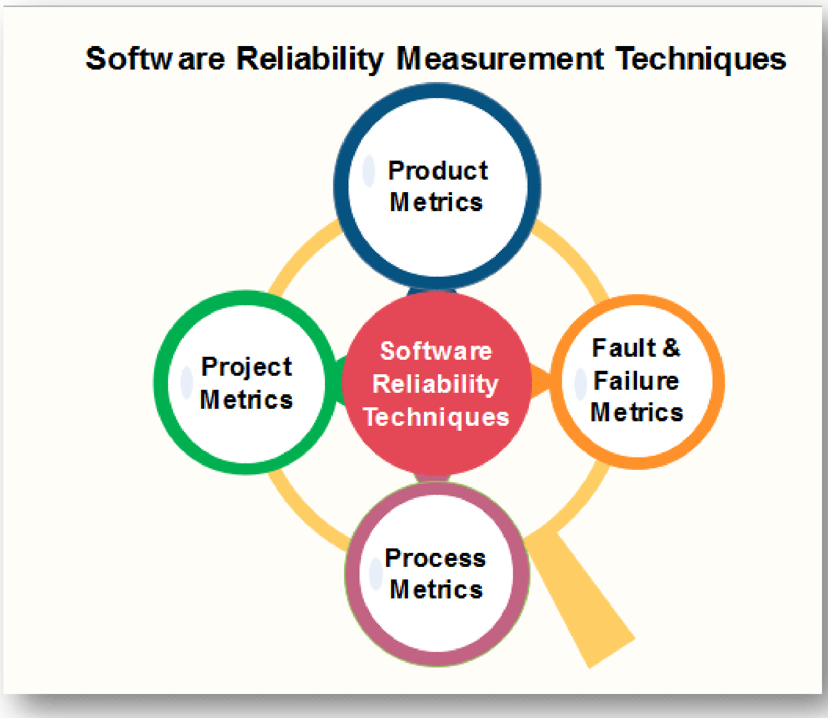
Software Reliability Measurement Techniques

Reliability metrics are used to quantitatively expressed the reliability of the software product.

The option of which parameter is to be used depends upon the type of system to which it applies & the requirements of the application domain.

The current methods of software reliability measurement can be divided into four categories:

The current methods of software reliability measurement can be divided into four categories:


Software Reliability Measurement Techniques
Product Metrics
Project Metrics
Software Reliability Techniques
Fault & Failure Metrics
Process Metrics

1. Product Metrics

Product metrics are those which are used to build the artifacts, i.e., requirement specification documents, system design documents, etc.

These metrics help in the assessment if the product is right sufficient through records on attributes like usability, reliability, maintainability & portability. In these measurements are taken from the actual body of the source code.

1. Software size is thought to be reflective of complexity, development effort, and reliability. Lines of Code (LOC), or LOC in thousands (KLOC), is an initial intuitive approach to measuring software size. The basis of LOC is that program length can be used as a predictor of program characteristics such as effort &ease of maintenance. It is a measure of the functional complexity of the program and is independent of the programming language.

2. Function point metric is a technique to measure the functionality of proposed software development based on the count of inputs, outputs, master files, inquires, and interfaces.

3. Test coverage metric size fault and reliability by performing tests on software products, assuming that software reliability is a function of the portion of software that is successfully verified or tested.

4. Complexity is directly linked to software reliability, so representing complexity is essential. Complexity-oriented metrics is a way of determining the complexity of a program's control structure by simplifying the code into a graphical representation. The representative metric is McCabe's Complexity Metric.

5. Quality metrics measure the quality at various steps of software product development. An vital quality metric is Defect Removal Efficiency (DRE). DRE provides a measure of quality because of different quality assurance and control activities applied throughout the development process.

2. Project Management Metrics

Project metrics define project characteristics and execution. If there is proper management of the project by the programmer, then this helps us to achieve better products. A relationship exists between the development process and the ability to complete projects on time and within the desired quality objectives. Cost increase when developers use inadequate methods. Higher reliability can be achieved by using a better development process, risk management process, configuration management process.

These metrics are:

1. Number of software developers
2. Staffing pattern over the life-cycle of the software
3. Cost and schedule
4. Productivity

3. Process Metrics

Process metrics quantify useful attributes of the software development process & its environment. They tell if the process is functioning optimally as they report on characteristics like cycle time & rework time. The goal of process metric is to do the right job on the first time through the process. The quality of the product is a direct function of the process. So process metrics can be used to estimate, monitor, and improve the reliability and quality of software. Process metrics describe the effectiveness and quality of the processes that produce the software product.

Examples are:

1. The effort required in the process
2. Time to produce the product
3. Effectiveness of defect removal during development
4. Number of defects found during testing

5, Maturity of the process

4. Fault and Failure Metrics

A fault is a defect in a program which appears when the programmer makes an error and causes failure when executed under particular conditions. These metrics are used to determine the failure-free execution software.

To achieve this objective, a number of faults found during testing and the failures or other problems which are reported by the user after delivery are collected, summarized, and analyzed. Failure metrics are based upon customer information regarding faults found after release of the software. The failure data collected is therefore used to calculate failure density, Mean Time between Failures (MTBF), or other parameters to measure or predict software reliability.

Reliability Growth Models

The reliability growth group of models measures and predicts the improvement of reliability programs through the testing process.

The growth model represents the reliability or failure rate of a system as a function of time or the number of test cases.

Models included in this group are as following below.

1. **Coutinho Model –** Coutinho adapted the Duane growth model to represent the software testing process. Coutinho plotted the cumulative number of deficiencies discovered and the number of correction actions made vs the cumulative testing weeks on log-log paper. Let N(t) denote the cumulative number of failures and let t be the total testing time. The failure rate, /\(t), the model can be expressed as

   /\(t) = N(t)/t**=B**ot^-B1

Where Bo & B1 are the model parameters. The least squares method can be used to estimate the parameters of this model.

2. Wall and Ferguson Model – Wall and Ferguson proposed a model similar to the Weibull growth model for predicting the failure rate of software during testing. The cumulative number of failures at time t, m(t), can be expressed as

$${m_{(t)} = a_{0}[b(t)]^{3}}$$

where alpha0 & alpha1 are the unknown parameters. The function b(t) can be obtained as the number of test cases or total testing time. Similarly, the failure rate function at time t is given by

$$\lambda_{(t)} = m^{'(t)} = a_{0}\beta b^{'(t)[b(t)]^{\beta-1}}$$

Wall and Ferguson tested this model using several software failure data and observed that failure data correlate well with the model.

There are several types of reliability growth models, including:

1. Non-homogeneous Poisson Process (NHPP) Model: This model is based on the assumption that the number of failures in a system follows a Poisson distribution. It is used to model the reliability growth of a system over time, and to predict the number of failures that will occur in the future.

2. Duane Model: This model is based on the assumption that the rate of failure of a system decreases over time as the system is improved. It is used to model the reliability growth of a system over time, and to predict the reliability of the system at any given time.

3. Gooitzen Model: This model is based on the assumption that the rate of failure of a system decreases over time as the system is improved, but that there may be periods of time where the rate of failure increases. It is used to model the reliability growth of a system over time, and to predict the reliability of the system at any given time.

4. Littlewood Model: This model is based on the assumption that the rate of failure of a system decreases over time as the system is improved, but that there may be periods of time where the rate of failure remains constant. It is used to model the reliability growth of a system over time, and to predict the reliability of the system at any given time.

Reliability growth models are useful tools for software engineers, as they can help to predict the reliability of a system over time and to guide the testing and improvement process. They can also help organizations to make informed decisions about the allocation of resources, and to prioritize improvements to the system.

Advantages of Reliability Growth Models:

1. Predicting Reliability: Reliability growth models are used to predict the reliability of a system over time, which can help organizations to make informed decisions about the allocation of resources and the prioritization of improvements to the system.

2. Guiding the Testing Process: Reliability growth models can be used to guide the testing process, by helping organizations to determine which tests should be run, and when they should be run, in order to maximize the improvement of the system's reliability.

3. Improving the Allocation of Resources: Reliability growth models can help organizations to make informed decisions about the allocation of resources, by providing an estimate of the expected reliability of the system over time, and by helping to prioritize improvements to the system.

4. Identifying Problem Areas: Reliability growth models can help organizations to identify problem areas in the system, and to focus their efforts on improving these areas in order to improve the overall reliability of the system.

Disadvantages of Reliability Growth Models:

1. Predictive Accuracy: Reliability growth models are only predictions, and actual results may differ from the predictions. Factors such as changes in the system, changes in the environment, and unexpected failures can impact the accuracy of the predictions.

2. Model Complexity: Reliability growth models can be complex, and may require a high level of technical expertise to understand and use effectively.

3. Data Availability: Reliability growth models require data on the system’s reliability, which may not be available or may be difficult to obtain.

Software Engineering Institute Capability Maturity Model (SEICMM)

The Capability Maturity Model (CMM) is a procedure used to develop and refine an organization's software development process.

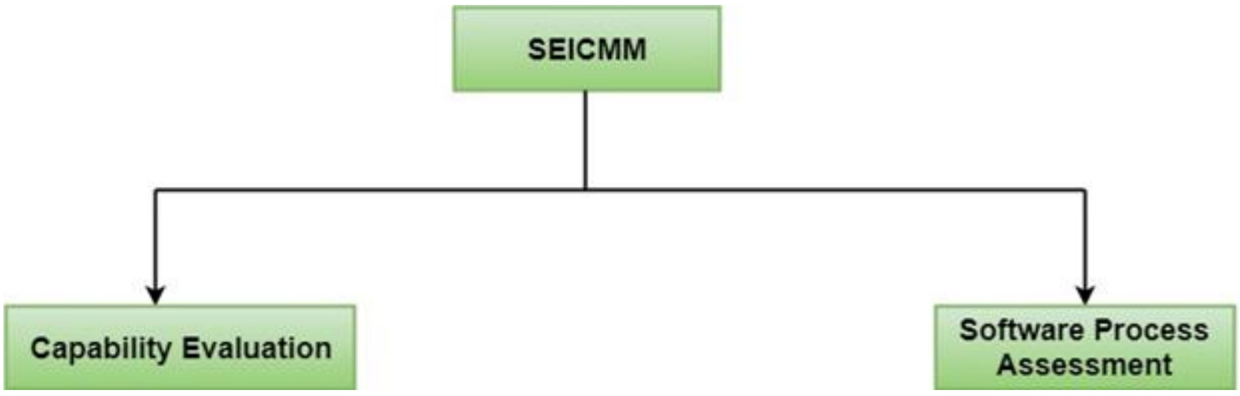
The model defines a five-level evolutionary stage of increasingly organized and consistently more mature processes.

CMM was developed and is promoted by the Software Engineering Institute (SEI), a research and development center promote by the U.S. Department of Defense (DOD).

Capability Maturity Model is used as a benchmark to measure the maturity of an organization's software process.

Methods of SEICMM

There are two methods of SEICMM:

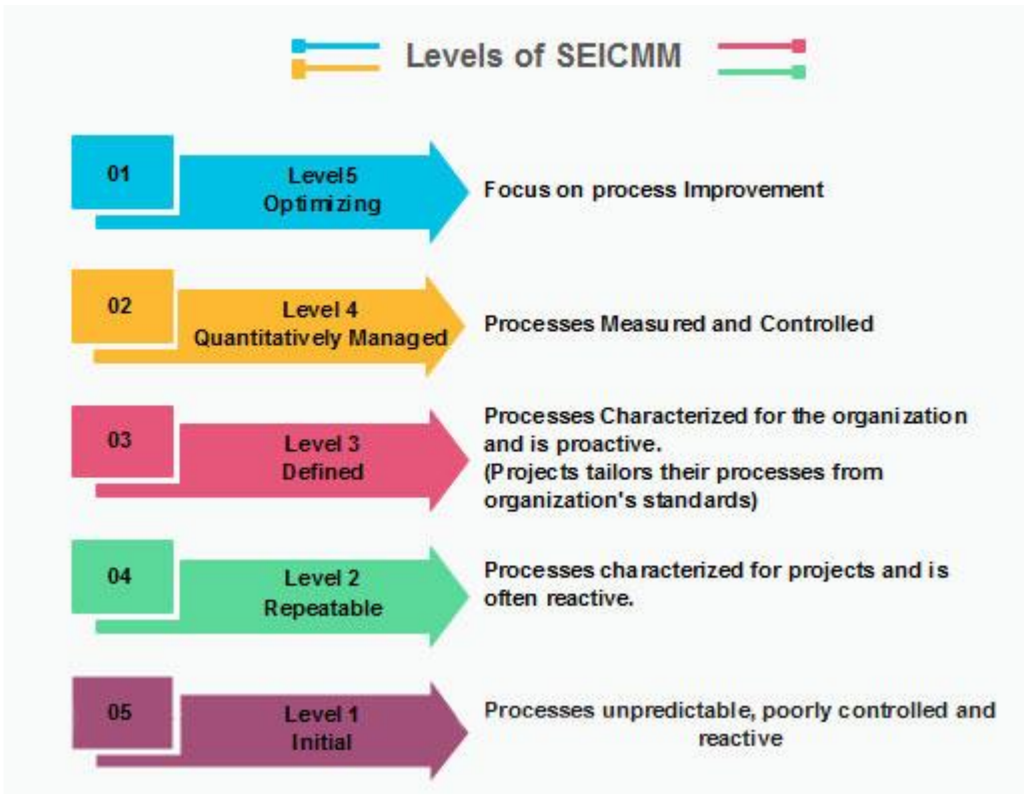


Software Engineering Institute Capability Maturity Model (SEICMM)

Capability Evaluation: Capability evaluation provides a way to assess the software process capability of an organization. The results of capability evaluation indicate the likely contractor performance if the contractor is awarded a work. Therefore, the results of the software process capability assessment can be used to select a contractor.

Software Process Assessment: Software process assessment is used by an organization to improve its process capability. Thus, this type of evaluation is for purely internal use.

SEI CMM categorized software development industries into the following five maturity levels. The various levels of SEI CMM have been designed so that it is easy for an organization to build its quality system starting from scratch slowly.


Levels of SEICMM
01
Level5 Optimizing
Focus on process Improvement
02
Level 4 Quantitatively Managed
Processes Measured and Controlled
03
Level 3 Defined
Processes Characterized for the organization and is proactive. (Projects tailors their processes from organization's standards)
04
Level 2 Repeatable
Processes characterized for projects and is often reactive.
05
Level 1 Initial
Processes unpredictable, poorly controlled and reactive

Software Engineering Institute Capability Maturity Model (SEICMM)

Level 1: Initial

Ad hoc activities characterize a software development organization at this level. Very few or no processes are described and followed. Since software production processes are not limited, different engineers follow their process and as a result, development efforts become chaotic. Therefore, it is also called a chaotic level.

Level 2: Repeatable

At this level, the fundamental project management practices like tracking cost and schedule are established. Size and cost estimation methods, like function point analysis, COCOMO, etc. are used.

Level 3: Defined

At this level, the methods for both management and development activities are defined and documented. There is a common organization-wide understanding of operations, roles, and responsibilities. The ways through defined, the process and product qualities are not measured. ISO 9000 goals at achieving this level.

Level 4: Managed

At this level, the focus is on software metrics. Two kinds of metrics are composed.

Product metrics measure the features of the product being developed, such as its size, reliability, time complexity, understandability, etc.

Process metrics follow the effectiveness of the process being used, such as average defect correction time, productivity, the average number of defects found per hour inspection, the average number of failures detected during testing per LOC, etc. The software process and product quality are measured, and quantitative quality requirements for the product are met. Various tools like Pareto charts, fishbone diagrams, etc. are used to measure the product and process quality. The process metrics are used to analyze if a project performed satisfactorily. Thus, the outcome of process measurements is used to calculate project performance rather than improve the process.

Level 5: Optimizing

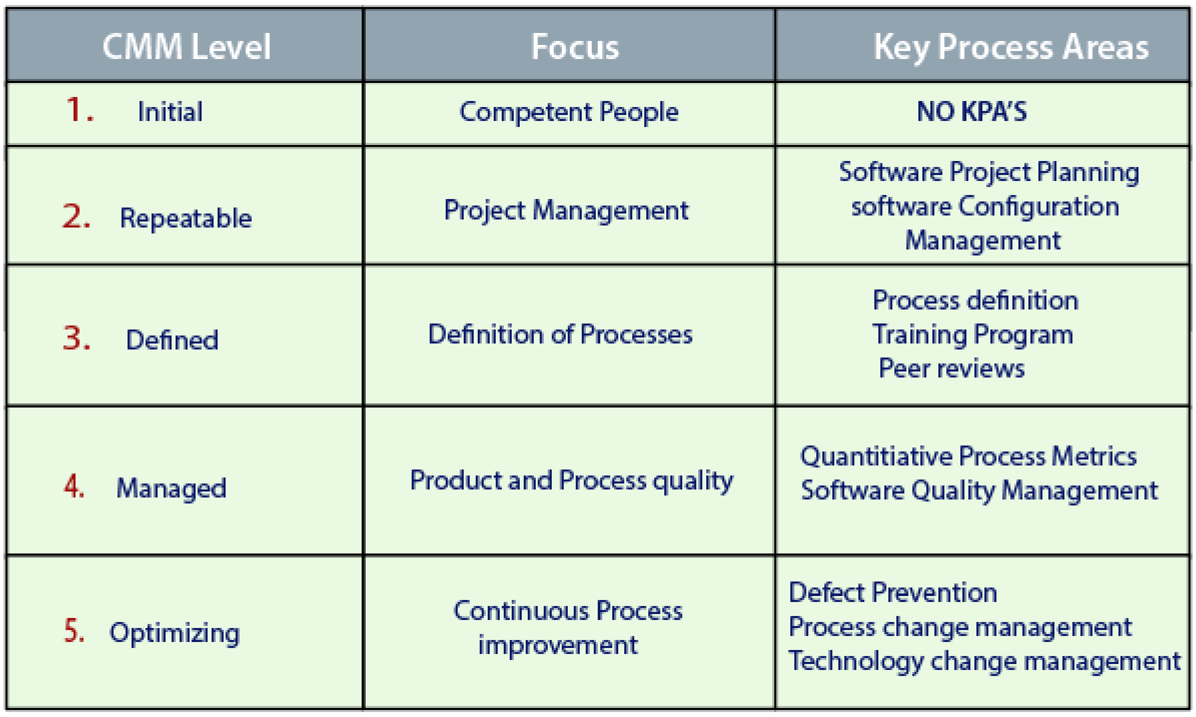
At this phase, process and product metrics are collected. Process and product measurement data are evaluated for continuous process improvement.

Key Process Areas (KPA) of a software organization

Except for SEI CMM level 1, each maturity level is featured by several Key Process Areas (KPAs) that contains the areas an organization should focus on improving its software process to the next level. The focus of each level and the corresponding key process areas are shown in the fig.

Software Engineering Institute Capability Maturity Model (SEICMM)

SEI CMM provides a series of key areas on which to focus to take an organization from one level of maturity to the next. Thus, it provides a method for gradual quality improvement over various stages. Each step has been carefully designed such that one step enhances the capability already built up.



CMM Level	Focus	Key Process Areas
1. Initial	Competent People	NO KPA'S
2. Repeatable	Project Management	Software Project Planning software Configuration Management
3. Defined	Definition of Processes	Process definition Training Program Peer reviews
4. Managed	Product and Process quality	Quantitiative Process Metrics Software Quality Management
5. Optimizing	Continuous Process improvement	Defect Prevention Process change management Technology change management

The focus of each SEI CMM level and the Corresponding Key process areas.

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 Reus

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 Re-Engineering

Software Re-Engineering is the examination and alteration of a system to reconstitute it in a new form.

The principle of Re-Engineering when applied to the software development process is called software re-engineering.

It positively affects software cost, quality, customer service, and delivery speed.

The re-Engineering procedure requires the following steps

1. Decide which components of the software we want to re-engineer. Is it the complete software or just some components of the software?

2. Do Reverse Engineering to learn about existing software functionalities.

3. Perform restructuring of source code if needed for example modifying functional-Oriented programs in Object-Oriented programs

4. Perform restructuring of data if required

5. Use Forward Engineering ideas to generate re-engineered software

The need for software Re-engineering: Software re-engineering is an economical process for software development and quality enhancement of the product. This process enables us to identify the useless consumption of deployed resources and the constraints that are restricting the development process so that the development process could be made easier and cost-effective (time, financial, direct advantage, optimize the code, indirect benefits, etc.) and maintainable. The software reengineering is necessary for having

a) Boost up productivity: Software reengineering increase productivity by optimizing the code and database so that processing gets faster.

b) Processes in continuity: The functionality of older software products can be still used while the testing or development of software.

c) Improvement opportunity: Meanwhile the process of software reengineering, not only software qualities, features, and functionality but also your skills are refined, and new ideas hit your mind. This makes the developer's mind accustomed to capturing new opportunities so that more and more new features can be developed.

d) Reduction in risks: Instead of developing the software product from scratch or from the beginning stage, developers develop the product from its existing stage to enhance some specific features brought in concern by stakeholders or its users. Such kind of practice reduces the chances of fault fallibility.

e) Saves time: As stated above, the product is developed from the existing stage rather than the beginning stage, so the time consumed in software engineering is lesser.

f) Optimization: This process refines the system features, and functionalities and reduces the complexity of the product by consistent optimization as maximum as possible.

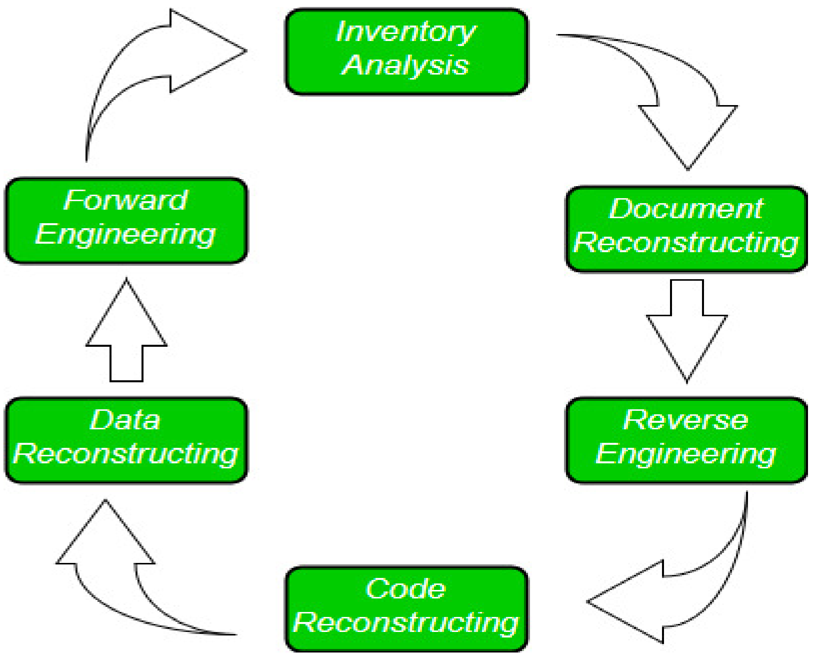
Re-Engineering cost factors:

1. The quality of the software is to be re-engineered.

2. The tool support availability for engineering.

3. The extent of the data conversion is required.

4. The availability of expert staff for Re-engineering.



Software Re-Engineering Activities:

1. Inventory Analysis:
Every software organization should have an inventory of all the applications.

. Inventory can be nothing more than a spreadsheet model containing information that provides a detailed description of every active application.

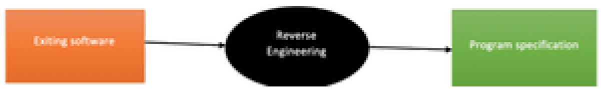
- By sorting this information according to business criticality, longevity, current maintainability, and other local important criteria, candidates for re-engineering appear.
- The resource can then be allocated to a candidate application for re-engineering work.

2. Document reconstructing:
Documentation of a system either explains how it operates or how to use it.

- Documentation must be updated.
- It may not be necessary to fully document an application.

- The system is business-critical and must be fully re-documented.

3. Reverse Engineering:
Reverse engineering is a process of design recovery. Reverse engineering tools extract data and architectural and procedural design information from an existing program.



4. Code Reconstructing:

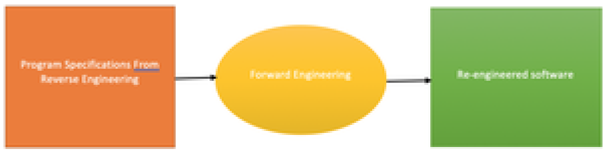
- To accomplish code reconstruction, the source code is analyzed using a reconstructing tool. Violations of structured programming construct are noted and code is then reconstructed.
- The resultant restructured code is reviewed and tested to ensure that no anomalies have been introduced.

5. Data Restructuring:

- Data restructuring begins with a reverse engineering activity.
- The current data architecture is dissected, and the necessary data models are defined.
- Data objects and attributes are identified, and existing data structures are reviewed for quality.

6. Forward Engineering:
Forward Engineering also called renovation or reclamation not only recovers design information from existing software but uses

this information to alter or reconstitute the existing system to improve its overall quality.



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;

1. Source code
2. Design and interfaces
3. User manuals
4. Software Documentation
5. Software requirement specifications and many more.

What are the advantages of software reuse?

1. Less effort: Software reuse requires less effort because many components use in the system are ready made components.

2. Time-saving: Re-using the ready made components is time saving for the software team.

3. Reduce cost: Less effort, and time saving leads to the overall cost reduction.

4. 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.

5. Utilize fewer resources: Software reuse save many sources just like effort, time, money etc.

6. Leads to a better quality software: Software reuse save our time and we can consume our more time on maintaining software quality and assurance.

Emerging topics

The 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 ?

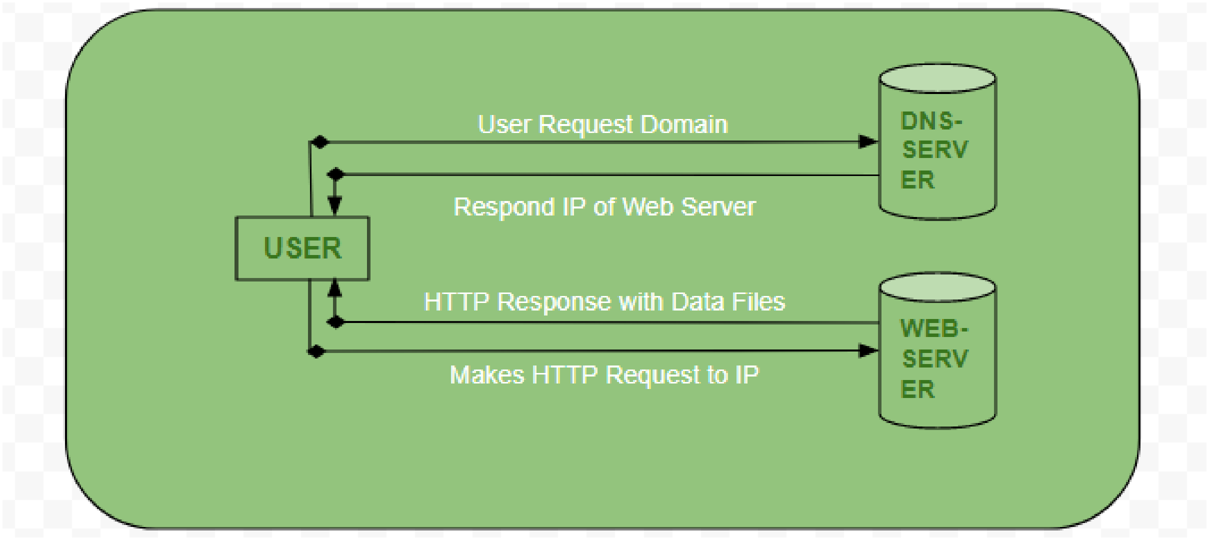
In this article we are going to take a dive into the Client-Server model and have a look at how the Internet works via, web browsers. This article will help us in having a solid foundation of the WEB and help in working with WEB technologies with ease.

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

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.

- 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.

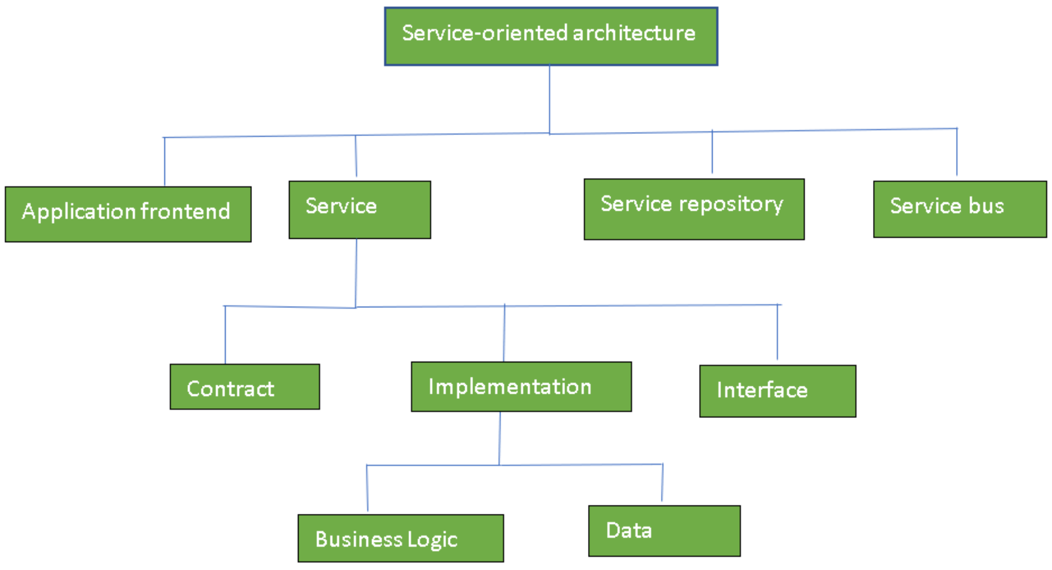
- SOA-based computing packages functionalities into a set of interoperable services, which can be integrated into different software systems belonging to separate business domains.

The different characteristics of SOA are as follows :
o Provides interoperability between the services.
o Provides methods for service encapsulation, service discovery, service composition,
service reusability and service integration.
o Facilitates QoS (Quality of Services) through service contract based on Service Level
Agreement (SLA).
o Provides loosely couples services.
o Provides location transparency with better scalability and availability.
o Ease of maintenance with reduced cost of application development and
deployment.
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

Software as a Service | SaaS

SaaS is also known as "**On-Demand Software**". It is a software distribution model in which services are hosted by a cloud service provider. These services are available to end-users over the internet so, the end-users do not need to install any software on their devices to access these services.

There are the following services provided by SaaS providers -

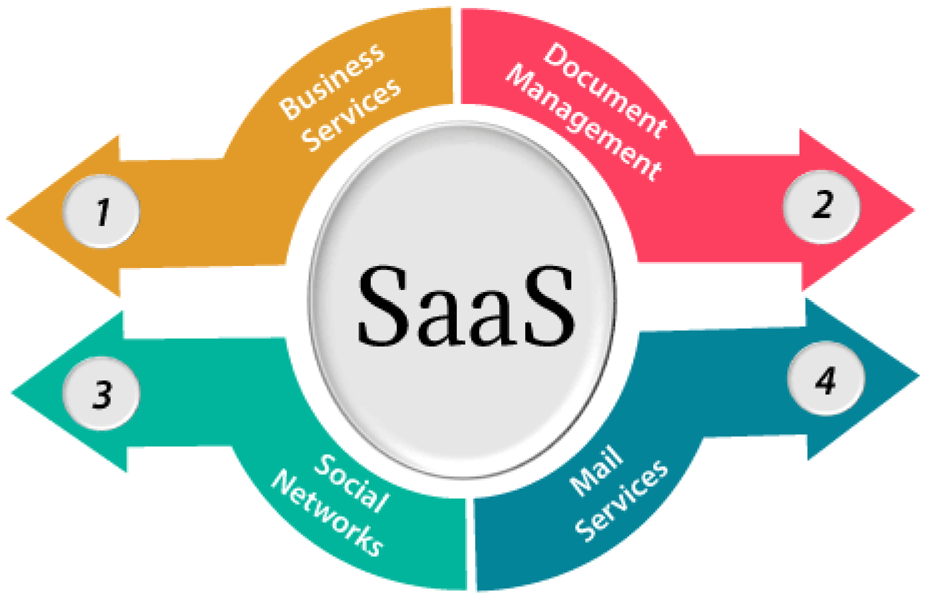
Business Services - SaaS Provider provides various business services to start-up the business. The SaaS business services include **ERP** (Enterprise Resource Planning), **CRM** (Customer Relationship Management), **billing**, and **sales**.

Document Management - SaaS document management is a software application offered by a third party (SaaS providers) to create, manage, and track electronic documents.

Example: Slack, Samepage, Box, and Zoho Forms.

Social Networks - As we all know, social networking sites are used by the general public, so social networking service providers use SaaS for their convenience and handle the general public's information.

Mail Services - To handle the unpredictable number of users and load on e-mail services, many e-mail providers offering their services using SaaS.



Advantages of SaaS cloud computing layer

1) SaaS is easy to buy

SaaS pricing is based on a monthly fee or annual fee subscription, so it allows organizations to access business functionality at a low cost, which is less than licensed applications.

Unlike traditional software, which is sold as a licensed based with an up-front cost (and often an optional ongoing support fee), SaaS providers are generally pricing the applications using a subscription fee, most commonly a monthly or annually fee.

2. One to Many

SaaS services are offered as a one-to-many model means a single instance of the application is shared by multiple users.

3. Less hardware required for SaaS

The software is hosted remotely, so organizations do not need to invest in additional hardware.

4. Low maintenance required for SaaS

Software as a service removes the need for installation, set-up, and daily maintenance for the organizations. The initial set-up cost for SaaS is typically less than the enterprise software. SaaS vendors are pricing their applications based on some usage parameters, such as a number of users using the application. So SaaS does easy to monitor and automatic updates.

5. No special software or hardware versions required

All users will have the same version of the software and typically access it through the web browser. SaaS reduces IT support costs by outsourcing hardware and software maintenance and support to the IaaS provider.

6. Multidevice support

SaaS services can be accessed from any device such as desktops, laptops, tablets, phones, and thin clients.

7. API Integration

SaaS services easily integrate with other software or services through standard APIs.

8. No client-side installation

SaaS services are accessed directly from the service provider using the internet connection, so do not need to require any software installation.

Disadvantages of SaaS cloud computing layer

1) Security

Actually, data is stored in the cloud, so security may be an issue for some users. However, cloud computing is not more secure than in-house deployment.

2) Latency issue

Since data and applications are stored in the cloud at a variable distance from the end-user, there is a possibility that there may be greater latency when interacting with the application compared to local deployment. Therefore, the SaaS model is not suitable for applications whose demand response time is in milliseconds.

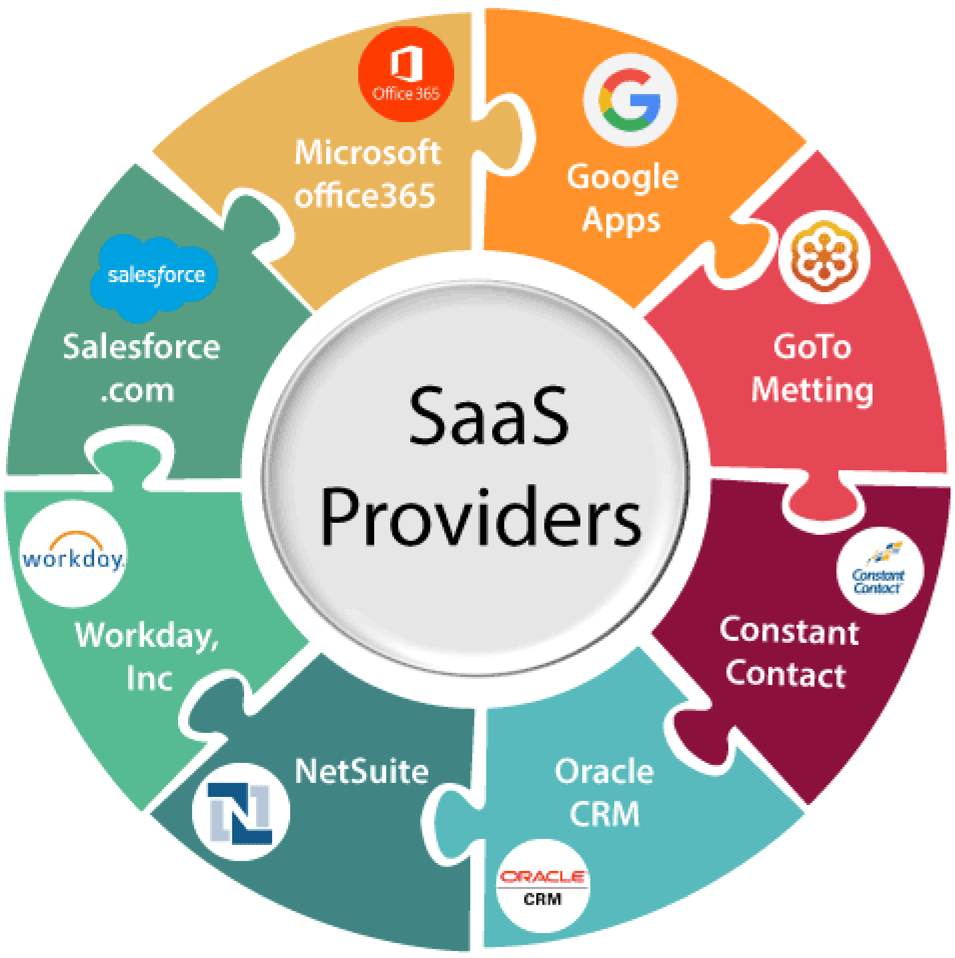
3) Total Dependency on Interne

Without an internet connection, most SaaS applications are not usable.

4) Switching between SaaS vendors is difficult

Switching SaaS vendors involves the difficult and slow task of transferring the very large data files over the internet and then converting and importing them into another SaaS also.

Popular SaaS Providers



The below table shows some popular SaaS providers and services that are provided by them -

Provider	**Services**
Salseforce.com	On-demand CRM solutions
Microsoft Office 365	Online office suite
Google Apps	Gmail, Google Calendar, Docs, and sites
NetSuite	ERP, accounting, order management, CRM, Professionals Services Automation

	(PSA), and e-commerce applications.
GoToMeeting	Online meeting and video-conferencing software
Constant Contact	E-mail marketing, online survey, and event marketing
Oracle CRM	CRM applications
Workday, Inc	Human capital management, payroll, and financial management.