UNIT-1

Introduction to Software Engineering: The evolving role of software, Changing Nature of Software, Software myths. Similarity and Differences from Conventional Engineering Processes, Software Quality Attributes. Software Development Life Cycle (SDLC)Models: Waterfall Model, Prototype Model, Spiral Model, Evolutionary Development Models, Iterative Enhancement Models.

INTRODUCTION:

Software Engineering is a framework for building software and is an engineering approach to software development. Software programs can be developed without S/E principles and methodologies but they are indispensable if we want to achieve good quality software in a cost-effective manner.

Software is defined as:
Instructions + Data Structures + Documents

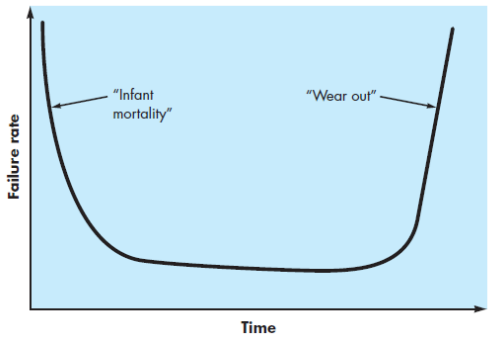
Engineering is the branch of science and technology concerned with the design, building, and use of engines, machines, and structures. It is the application of science, tools and methods to find cost effective solution to simple and complex problems.
SOFTWARE ENGINEERING is defined as a systematic, disciplined and quantifiable approach for the development, operation and maintenance of software.

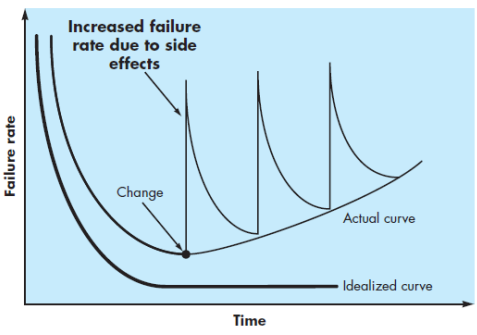
The Evolving role of software:

The dual role of Software is as follows:
1. A Product- Information transformer producing, managing and displaying information.
2. A Vehicle for delivering a product- Control of computer(operating system),the communication of
information(networks) and the creation of other programs.

Characteristics of software

- **Software is developed or engineered**, but it is not manufactured in the classical sense.
- **Software does not wear out,** but it deteriorates due to change.
- **Software is custom built** rather than assembling existing components.





Although the industry is moving towards component-based construction, most software continues to be custom built

THE CHANGING NATURE OFSOFTWARE

The various categories of software are
1. System software
2. Application software
3. Engineering and scientific software
4. Embedded software
5. Product-line software
6. Web-applications
7. Artificial intelligence software

• **System software.** System software is a collection of programs written to service other programs
• **Embedded software**-- resides in read-only memory and is used to control products and systems for the consumer and industrial markets.
• **Artificial intelligence software.** Artificial intelligence (AI) software makes use of nonnumeric algorithms to solve complex problems that are not amenable to computation or straightforward analysis
Engineering and scientific software. Engineering and scientific software have been characterized by "number crunching" algorithms.

LEGACY SOFTWARE

Legacy software are older programs that are developed decades ago. The quality of legacy software is poor because it has inextensible design, convoluted code, poor and non-existent documentation, test cases and results that are not achieved.
As time passes legacy systems evolve due to following reasons:
The software must be adapted to meet the needs of new computing environment or technology.
The software must be enhanced to implement new business requirements.
The software must be extended to make it interoperable with more modern systems or database
The software must be rearchitected to make it viable within a network environment.

SOFTWARE APPLICATIONS

System Software –
System Software is necessary to manage the computer resources and support the execution of application programs. Software like operating systems, compilers, editors and drivers, etc., come under this category. A computer cannot function without the presence of these. Operating systems are needed to link the machine-dependent needs of a program with the

capabilities of the machine on which it runs. Compilers translate programs from high-level language to machine language.

Application Software –
Application software is designed to fulfil the user's requirement by interacting with the user directly. It could be classified into two major categories: - generic or customized. Generic Software is the software that is open to all and behaves the same for all of its users. Its function is limited and not customized as per the changing requirements of the user. However, on the other hand, Customized software the software products which are designed as per the client's requirement, and are not available for all.

Networking and Web Applications Software –
Networking Software provides the required support necessary for computers to interact with each other and with data storage facilities. The networking software is also used when software is running on a network of computers (such as the World Wide Web). It includes all network management software, server software, security and encryption software, and software to develop web-based applications like HTML, PHP, XML, etc.

Embedded Software –
This type of software is embedded into the hardware normally in the Read-Only Memory (ROM) as a part of a large system and is used to support certain functionality under the control conditions. Examples are software used in instrumentation and control applications like washing machines, satellites, microwaves, etc.

Reservation Software –
A Reservation system is primarily used to store and retrieve information and perform transactions related to air travel, car rental, hotels, or other activities. They also provide access to bus and railway reservations, although these are not always integrated with the main system. These are also used to relay computerized information for users in the hotel industry, making a reservation and ensuring that the hotel is not overbooked.

Business Software –
This category of software is used to support business applications and is the most widely used category of software. Examples are software for inventory management, accounts, banking, hospitals, schools, stock markets, etc.

Entertainment Software –
Education and entertainment software provides a powerful tool for educational agencies, especially those that deal with educating young children. There is a wide range of entertainment software such as computer games, educational games, translation software, mapping software, etc.

Artificial Intelligence Software –
Software like expert systems, decision support systems, pattern recognition software, artificial neural networks, etc. come under this category. They involve complex problems which are not affected by complex computations using non-numerical algorithms.

Scientific Software –
Scientific and engineering software satisfies the needs of a scientific or engineering user to perform enterprise-specific tasks. Such software is written for specific applications using

principles, techniques, and formulae specific to that field. Examples are software like MATLAB, AUTOCAD, PSPICE, ORCAD, etc.

Utilities Software –
The programs coming under this category perform specific tasks and are different from other software in terms of size, cost, and complexity. Examples are anti-virus software, voice recognition software, compression programs, etc.

Document Management Software –
Document Management Software is used to track, manage and store documents in order to reduce the paperwork. Such systems are capable of keeping a record of the various versions created and modified by different users (history tracking). They commonly provide storage, versioning, metadata, security, as well as indexing and retrieval capabilities.

SOFTWARE MYTHS

Erroneous beliefs about software and the process that is used to build it can be traced to the earliest days of computing. Myths have several attributes that make them insidious. For instance, they appear to be reasonable statements of fact (sometimes containing elements of truth), have an intuitive feel, and are often promulgated by experienced practitioners who "know the score."

Today, most knowledgeable software engineering professionals recognize myths for what they are misleading attitudes that have caused serious problems for managers and practitioners alike. However, old attitudes and habits are difficult to modify, and remnants of software myths remain.

Three types of myths

1) Management myths

2) Customer myths

3) Practitioner myths

Software myths: erroneous beliefs about software and the process that is used to build it—can be traced to the earliest days of computing. Myths have a number of attributes that make them insidious. For instance, they appear to be reasonable statements of fact (sometimes containing elements of truth), they have an intuitive feel, and they are often promulgated by experienced practitioners who "know the score."

Today, most knowledgeable software engineering professionals recognize myths for what they are misleading attitudes that have caused serious problems for managers and practitioners alike. However, old attitudes and habits are difficult to modify, and remnants of software myths remain. Management myths. Managers with software responsibility, like managers in most disciplines, are often under pressure to maintain budgets, keep schedules from slipping, and improve quality. Like a drowning person who grasps at a straw, a software manager often grasps at belief in a software myth, if that belief will lessen the pressure (even temporarily).

Myth: We already have a book that's full of standards and procedures for building software. Won't that provide my people with everything they need to know?

Reality: The book of standards may very well exist, but is it used? Are software practitioners aware of its existence? Does it reflect modern software engineering practice? Is it complete? Is it adaptable? Is it streamlined to improve time-to-delivery while still maintaining a focus on quality? In many cases, the answer to all of these questions is "no."

Myth: If we get behind schedule, we can add more programmers and catch up (sometimes called the "Mongolian horde" concept).

Reality: Software development is not a mechanistic process like manufacturing. In the words of Brooks [Bro95]: "adding people to a late software project makes it later." At first, this statement may seem counterintuitive. However, as new people are added, people who were working must spend time educating the newcomers, thereby reducing the amount of time spent on productive development effort. People can be added but only in a planned and well-coordinated manner.

Myth: If I decide to outsource the software project to a third party, I can just relax and let that firm build it.

Reality: If an organization does not understand how to manage and control software projects internally, it will invariably struggle when it outsources software projects.

Customer myths:

A customer who requests computer software may be a person at the next desk, a technical group down the hall, the marketing/sales department, or an outside company that has requested software under contract. In many cases, the customer believes myths about software because software managers and practitioners do little to correct misinformation. Myths lead to false expectations (by the customer) and, ultimately, dissatisfaction with the developer.

Myth: A general statement of objectives is sufficient to begin writing programs—we can fill in the details later

Reality: Although a comprehensive and stable statement of requirements is not always possible, an ambiguous "statement of objectives" is a recipe for disaster. Unambiguous requirements (usually derived iteratively) are developed only through effective and continuous communication between customer and developer.

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

Reality: It is true that software requirements change, but the impact of change varies with the time at which it is introduced. When requirements changes are requested early (before design or code has been started), the cost impact is relatively small.16 However, as time passes, the cost impact grows rapidly—resources have been committed, a design framework has been established, and change can cause upheaval that requires additional resources and major design modification.

Practitioner's myths:

Myths that are still believed by software practitioners have been fostered by over 50 years of programming culture. During the early days, programming was viewed as an art form. Old ways and attitudes die hard.

Myth: Once we write the program and get it to work, our job is done.

Reality: Someone once said that "the sooner you begin 'writing code,' the longer it'll take you to get done." Industry data indicate that between 60 and 80 percent of all effort expended on software will be expended after it is delivered to the customer for the first time.

Myth: Until I get the program "running" I have no way of assessing its quality.

Reality: One of the most effective software quality assurance mechanisms can be applied from the inception of a project—the technical review. Software reviews (described in Chapter 15) are a "quality filter" that have been found to be more effective than testing for finding certain classes of software defects.

Myth: The only deliverable work product for a successful project is the working program.

Reality: A working program is only one part of a software configuration that includes many elements. A variety of work products (e.g., models, documents, plans) provide a foundation for successful engineering and, more important, guidance for software support.

Myth: Software engineering will make us create voluminous and unnecessary documentation and will invariably slow us down.

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

Many software professionals recognize the fallacy of the myths just described. Regrettably, habitual attitudes and methods foster poor management and technical practices, even when reality dictates a better approach. Recognition of software realities is the first step toward formulation of practical solutions for software engineering.

Similarity and Differences from Conventional Engineering Processes

Software Engineering Process and Conventional Engineering Process, both are processes related to computers and development

Software Engineering Process is an engineering process that is mainly related to computers and programming and developing different kinds of applications through the use of information technology.

Conventional Engineering Process is an engineering process that is highly based on empirical knowledge and is about building cars, machines, and hardware. It is a process that mainly involves science, mathematics, etc.

Both Software Engineering and Conventional Engineering Processes become automated after some time.

Both these processes are making our day-to-day place better.

Both these processes have a fixed working time.

Both processes must consist of deeper knowledge.

Software Engineering Process	Conventional Engineering Process
Software Engineering Process is a process that majorly involves computer science, information technology, and discrete mathematics.	The conventional Engineering Process is a process that majorly involves science, mathematics, and empirical knowledge.
It is mainly related to computers, programming, and writing codes for building applications.	It is about building cars, machines, hardware, buildings, etc.
In Software Engineering Process construction and development cost is low.	In Conventional Engineering Process construction and development cost is high.
It can involve the application of new and untested elements in software projects.	It usually applies only known and tested principles to meet product requirements.
In Software Engineering Process, most development effort goes into building new designs and features.	In Conventional Engineering Process, most development efforts are required to change old designs.
It majorly emphasizes quality.	It majorly emphasizes mass production.
Product development develops intangible products (software).	Product development develops tangible products (e.g. bridges, buildings).
Design requirements may change throughout the development process.	Design Requirements are typically well-defined upfront.

It can involve the application of new and untested elements in software projects.	It usually applies only known and tested principles to meet product requirements.
In Software Engineering Process, most development effort goes into building new designs and features.	In Conventional Engineering Process, most development efforts are required to change old designs.
It majorly emphasizes quality.	It majorly emphasizes mass production.
Product development develops intangible products (software).	Product development develops tangible products (e.g. bridges, buildings).
Design requirements may change throughout the development process.	Design Requirements are typically well-defined upfront.

Software Quality Attributes

The various factors, which influence the software, are termed as software factors. They can be broadly divided into two categories. The first category of the factors is of those that can be measured directly such as the number of logical errors, and the second category clubs those factors which can be measured only indirectly. For example, maintainability but each of the factors is to be measured to check for the content and the quality control.

Several models of software quality factors and their categorization have been suggested over the years. The classic model of software quality factors, suggested by McCall, consists of 11 factors (McCall et al., 1977). Similarly, models consisting of 12 to 15 factors were suggested by Deutsch and Willis (1988) and by Evans and Marciniak (1987).

All these models do not differ substantially from McCall's model. The McCall factor model provides a practical, up-to-date method for classifying software requirements (Pressman, 2000).

This model classifies all software requirements into 11 software quality factors. The 11 factors are grouped into three categories – product operation, product revision, and product transition factors.

- **Product operation factors** – Correctness, Reliability, Efficiency, Integrity, Usability.
- **Product revision factors** – Maintainability, Flexibility, Testability.
- **Product transition factors** – Portability, Reusability, Interoperability.

Product Operation Software Quality Factors

According to McCall's model, product operation category includes five software quality factors, which deal with the requirements that directly affect the daily operation of the software. They are as follows −

Correctness

These requirements deal with the correctness of the output of the software system. They include −

- Output mission
- The required accuracy of output that can be negatively affected by inaccurate data or inaccurate calculations.
- The completeness of the output information, which can be affected by incomplete data.
- The up-to-datedness of the information defined as the time between the event and the response by the software system.
- The availability of the information.
- The standards for coding and documenting the software system.

Reliability

Reliability requirements deal with service failure. They determine the maximum allowed failure rate of the software system, and can refer to the entire system or to one or more of its separate functions.

Efficiency

It deals with the hardware resources needed to perform the different functions of the software system. It includes processing capabilities (given in MHz), its storage capacity (given in MB or GB) and the data communication capability (given in MBPS or GBPS).

It also deals with the time between recharging of the system's portable units, such as, information system units located in portable computers, or meteorological units placed outdoors.

Integrity

This factor deals with the software system security, that is, to prevent access to unauthorized persons, also to distinguish between the group of people to be given read as well as write permit.

Usability

Usability requirements deal with the staff resources needed to train a new employee and to operate the software system.

Product Revision Quality Factors

According to McCall's model, three software quality factors are included in the product revision category. These factors are as follows −

Maintainability

This factor considers the efforts that will be needed by users and maintenance personnel to identify the reasons for software failures, to correct the failures, and to verify the success of the corrections.

Flexibility

This factor deals with the capabilities and efforts required to support adaptive maintenance activities of the software. These include adapting the current software to additional circumstances and

customers without changing the software. This factor's requirements also support perfective maintenance activities, such as changes and additions to the software in order to improve its service and to adapt it to changes in the firm's technical or commercial environment.

Testability

Testability requirements deal with the testing of the software system as well as with its operation. It includes predefined intermediate results, log files, and also the automatic diagnostics performed by the software system prior to starting the system, to find out whether all components of the system are in working order and to obtain a report about the detected faults. Another type of these requirements deals with automatic diagnostic checks applied by the maintenance technicians to detect the causes of software failures.

Product Transition Software Quality Factor

According to McCall's model, three software quality factors are included in the product transition category that deals with the adaptation of software to other environments and its interaction with other software systems. These factors are as follows −

Portability

Portability requirements tend to the adaptation of a software system to other environments consisting of different hardware, different operating systems, and so forth. The software should be possible to continue using the same basic software in diverse situations.

Reusability

This factor deals with the use of software modules originally designed for one project in a new software project currently being developed. They may also enable future projects to make use of a given module or a group of modules of the currently developed software. The reuse of software is expected to save development resources, shorten the development period, and provide higher quality modules.

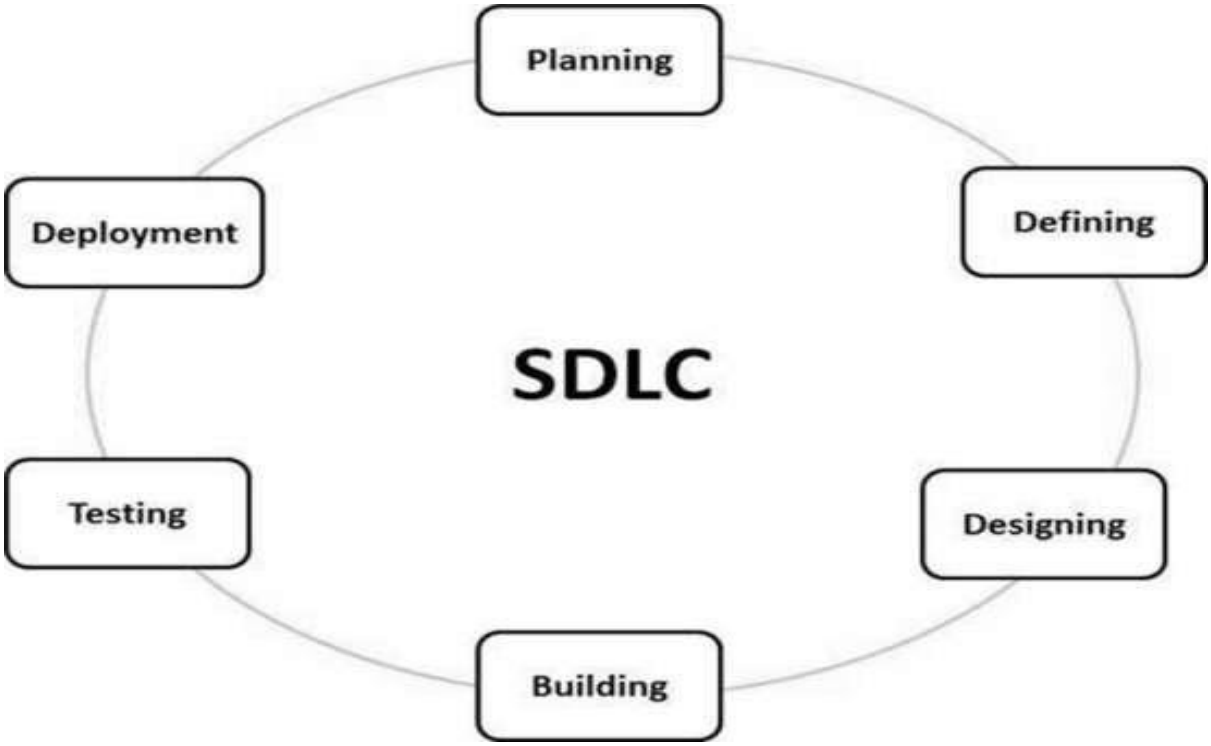
Interoperability

Interoperability requirements focus on creating interfaces with other software systems or with other equipment firmware. For example, the firmware of the production machinery and testing equipment interfaces with the production control software.

Software Development Life Cycle (SDLC)Models

SDLC stands for Software Development Life Cycle. SDLC is a process that consists of a series of planned activities to develop or alter the Software Products.

- Software Development Life Cycle (SDLC) is a process used by the software industry to design, develop and test high quality software's. The SDLC aims to produce a high-quality software that meets or exceeds customer expectations, reaches completion within times and cost estimates.
- SDLC is a process followed for a software project, within a software organization. It consists of a detailed plan describing how to develop, maintain, replace and alter or enhance specific software. The life cycle defines a methodology for improving the quality of software and the overall development process.



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

Stage 1: Planning and Requirement Analysis

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

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

Stage 2: Defining Requirements

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

Stage 3: Designing the Product Architecture

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

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

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

Stage 4: Building or Developing the Product

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

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

Stage 5: Testing the Product

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

Stage 6: Deployment in the Market and Maintenance

Once the product is tested and ready to be deployed it is released formally in the appropriate market. Sometimes product deployment happens in stages as per the business strategy of that organization. The product may first be released in a limited segment and tested in the real business environment (UAT- User acceptance testing).

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

SDLC Models

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

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

- Waterfall Model
- Prototype Model,
- Spiral Model,
- Evolutionary Development Models,
- Iterative Enhancement Models.

Water fall Model:

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

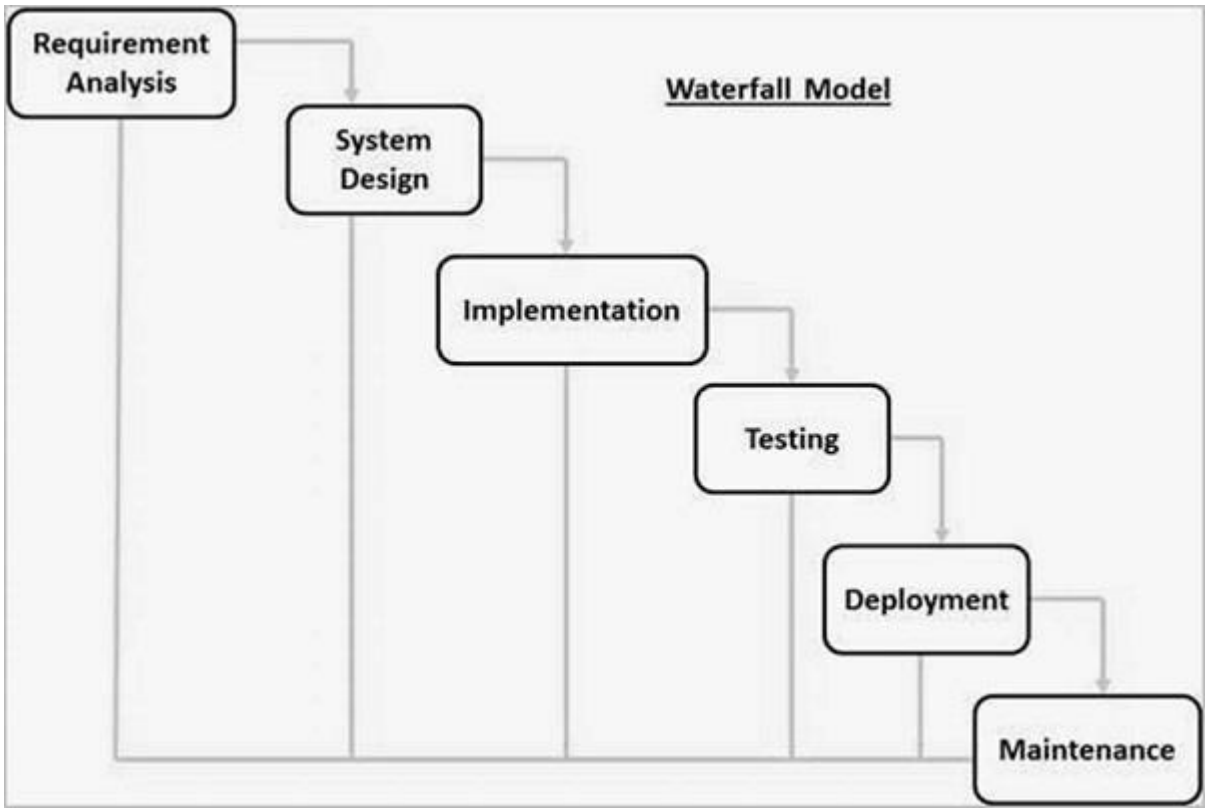
The Waterfall model is the earliest SDLC approach that was used for software development.

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

Waterfall Model - Design

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

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



The sequential phases in Waterfall model are −

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

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

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

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

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

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

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

Waterfall Model - Application

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

Requirements are very well documented, clear and fixed.

Product definition is stable.

Technology is understood and is not dynamic.

There are no ambiguous requirements.

Ample resources with required expertise are available to support the product.

The project is short.

Waterfall Model - Advantages

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

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

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

- Simple and easy to understand and use
- Easy to manage due to the rigidity of the model. Each phase has specific deliverables and a review process.
- Phases are processed and completed one at a time.
- Works well for smaller projects where requirements are very well understood.
- Clearly defined stages.
- Well understood milestones.
- Easy to arrange tasks.
- Process and results are well documented.

Waterfall Model - Disadvantages

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

The major disadvantages of the Waterfall Model are as follows −

No working software is produced until late during the life cycle.

High amounts of risk and uncertainty.

Not a good model for complex and object-oriented projects.

Poor model for long and ongoing projects.

Not suitable for the projects where requirements are at a moderate to high risk of changing. So, risk and uncertainty is high with this process model.

It is difficult to measure progress within stages.

Cannot accommodate changing requirements.

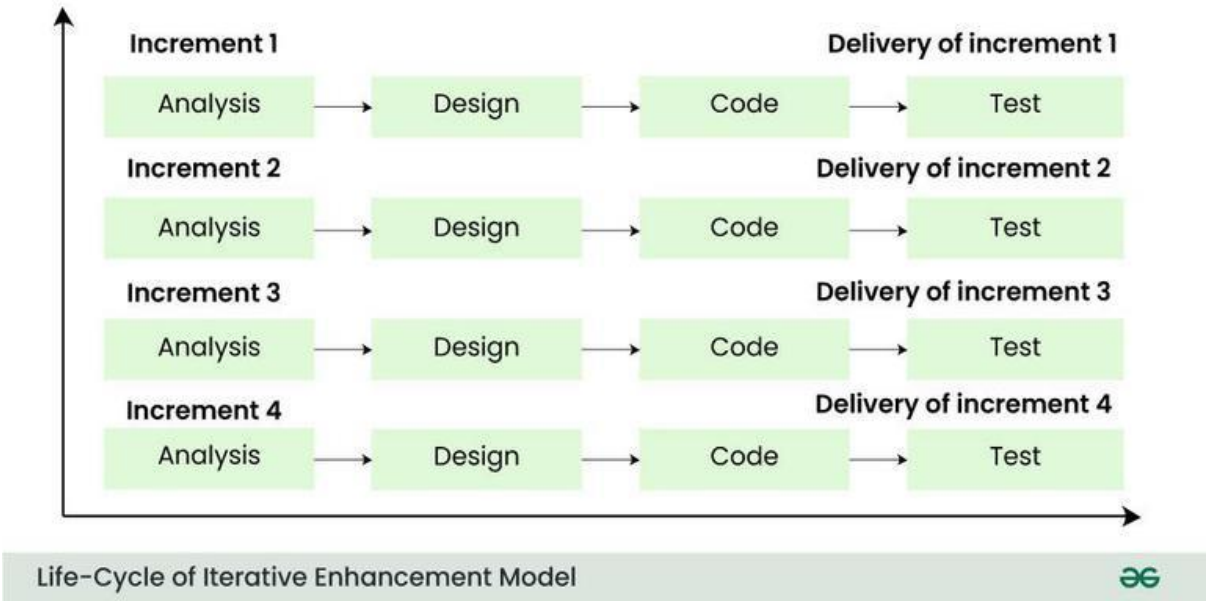
Adjusting scope during the life cycle can end a project.

Integration is done as a "big-bang. at the very end, which doesn't allow identifying any technological or business bottleneck or challenges early.

Iterative Enhancement Models

Software development uses a dynamic and adaptable method called the iterative enhancement Model. The iterative enhancement model encourages a software product's ongoing evolution and improvement. This methodology is noticeable due to its concentration on adaptability, flexibility and change responsiveness. It makes it easier for a product to evolve because it gives developers the freedom to progressively enhance the software, making sure that it complies with evolving specifications, user demands, and market demands. This helps products evolve more easily.

The Iterative Enhancement Model creates an environment where development teams can more effectively adjust to changing requirements by segmenting the software development process into smaller, more manageable parts. Every iteration improves on the one before it, adding new features and fixing problems found in earlier stages.



Life-Cycle of Iterative Enhancement Model

Applicability

Mobile app development: Updates and improvements are often needed for mobile apps to stay current with new devices, operating system versions and user preferences. By using an iterative process developers can release the beta versions of their apps, get user feedback and then improve functionality of those iterations in future release.

Web Application Development: The requirements for developing web applications frequently change as a result of shifting user demand and advancements in technology. The Iterative Enhancement Model makes it possible to developed features incrementally and guaranteeing that the application can be modified to satisfy changing user and market demands. In later iterations it also makes it easier to incorporate new features based on input from users.

E-commerce Platforms: Development in e-commerce field often involves constant updates. Implementing an iterative approach enables the introduction of new functionality.

Advantages of Iterative Enhancement Model

Adaptation to changing requirements is made possible by its flexibility in accommodating modifications and improvement during each iteration.

Problems and risks can be identified and addressed early in the development process, reduces chances of issue for future stages.

Every iteration is put through testing and improvement, leading to higher quality product.

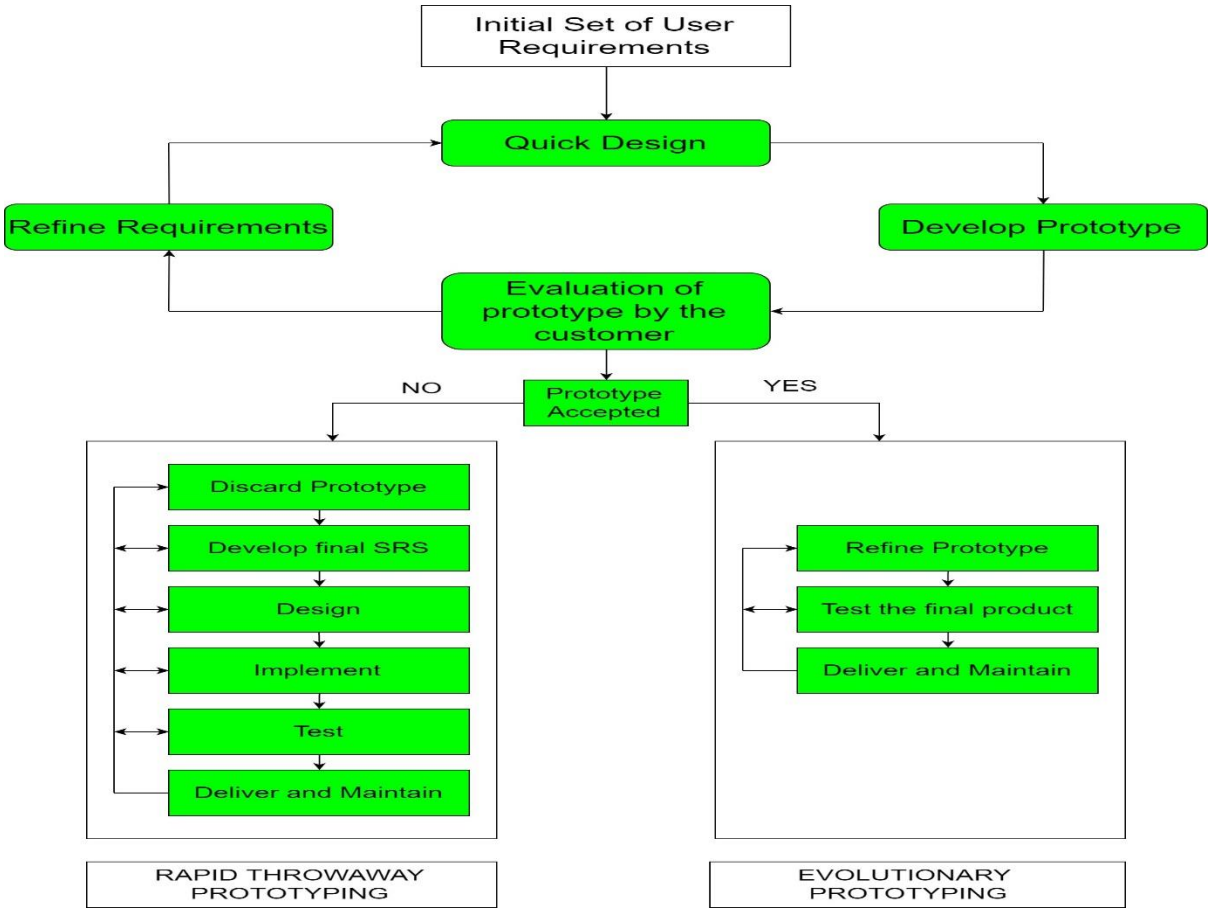
Disadvantages of Iterative Enhancement Model

Especially in larger projects, managing several iterations at once can add complexity.

Higher cost Due to constant changes, there may be delays in documentation,

Prototype Model

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



Prototype Model

Steps of Prototype Model

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

Advantage of Prototype Model

1. Reduce the risk of incorrect user requirement

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

Disadvantage of Prototype Model

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

Spiral Model

The Spiral Model. Originally proposed by Barry Boehm [Boe88], the spiral model is an evolutionary software process model that couples the iterative nature of prototyping with the controlled and systematic aspects of the waterfall model. It provides the potential for rapid development of increasingly more complete versions of the software. Boehm [Boe01a] describes the model in the following manner.

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

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

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

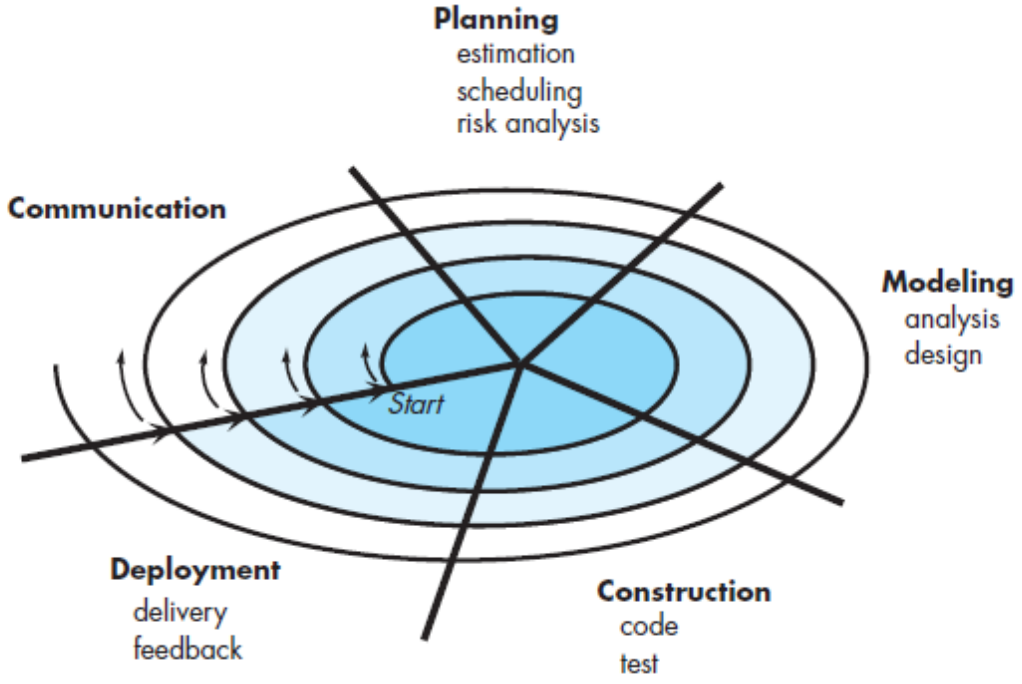


Figure. A typical spiral model

The first circuit around the spiral might result in the development of a product specification; subsequent passes around the spiral might be used to develop a prototype and then progressively more sophisticated versions of the software. Each pass through the planning region results in adjustments to the project plan. Cost and schedule are adjusted based on feedback derived from the customer after delivery. In addition, the project manager adjusts the planned number of iterations required to complete the software.

Unlike other process models that end when software is delivered, the spiral model can be adapted to apply throughout the life of the computer software. Therefore, the first circuit around the spiral might represent a "concept development project" that starts at the core of the spiral and continues for multiple iterations 5 until concept development is complete. If the concept is to be developed into an actual product, the process proceeds outward on the spiral and a "new product development project" commences. The new product will evolve through a number of iterations around the spiral. Later, a circuit around the spiral might be used to represent a "product enhancement project." In essence, the spiral, when characterized in this way, remains operative until the software is retired. There are times when the process is dormant, but whenever a change is initiated, the process starts at the appropriate entry point (e.g., product enhancement).

The spiral model is a realistic approach to the development of large-scale systems and software. Because software evolves as the process progresses, the developer and customer better understand and react to risks at each evolutionary level. The spiral model uses prototyping as a risk reduction mechanism but, more important, enables you to apply the prototyping approach at any stage in the evolution of the product. It maintains the systematic stepwise approach suggested by the classic life cycle but incorporates it into an iterative framework that more realistically reflects the real world. The spiral model demands a direct consideration of technical risks at all stages of the project and, if properly applied, should reduce risks before they become problematic.

But like other paradigms, the spiral model is not a panacea. It may be difficult to convince customers (particularly in contract situations) that the evolutionary approach is controllable. It demands considerable risk assessment expertise and relies on this expertise for success. If a major risk is not uncovered and managed, problems will undoubtedly occur.

When to use Spiral Model?

- When deliverance is required to be frequent.
- When the project is large
- When requirements are unclear and complex
- When changes may require at any time
- Large and high budget projects
-

Advantages

- High amount of risk analysis
- Useful for large and mission-critical projects.

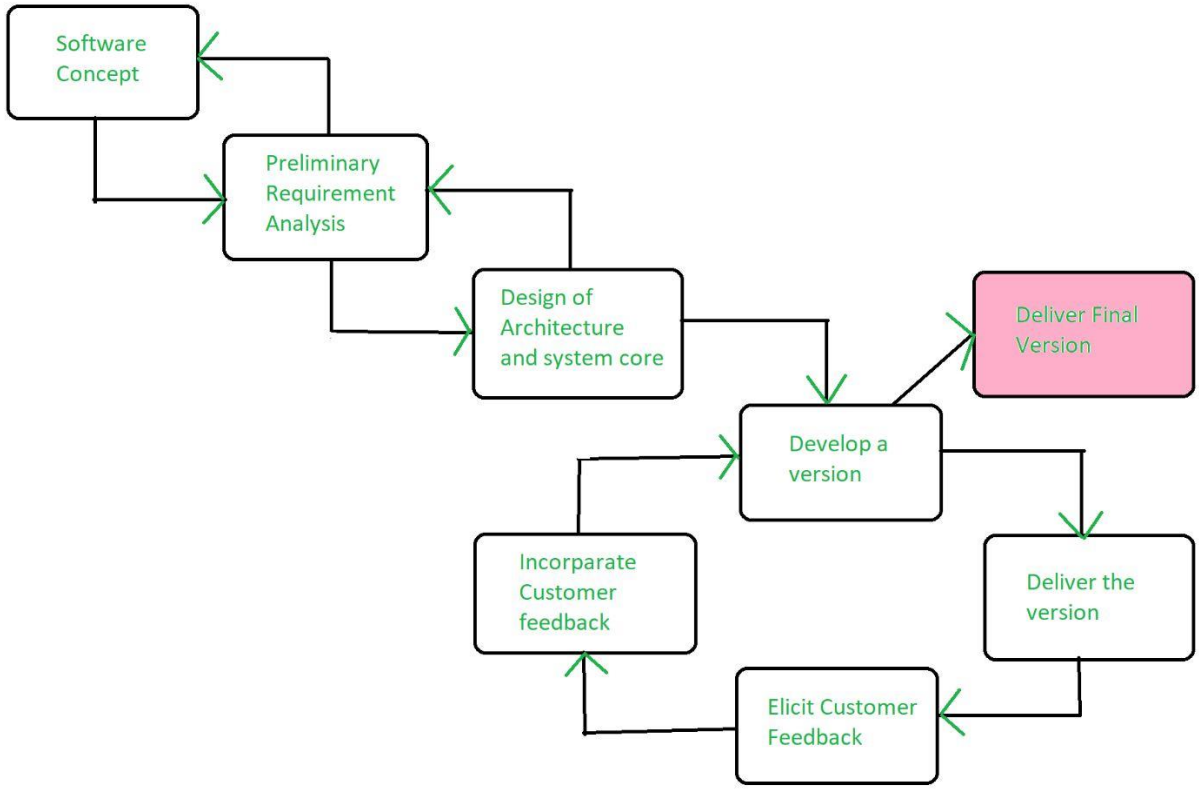
Disadvantages

- Can be a costly model to use.
- Risk analysis needed highly particular expertise
- Doesn't work well for smaller projects.
-

Evolutionary Development Models:

The evolutionary model is a combination of the Iterative and Incremental models of the software development life cycle. Delivering your system in a big bang release, delivering it in incremental process over time is the action done in this model. Some initial requirements and architecture envisioning need to be done. It is better for software products that have their feature sets redefined during development because of user feedback and other factors. This article focuses on discussing the Evolutionary Model in detail.

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



Application of Evolutionary Model

- 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.
- 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

- Customer needs are clear and been explained in deep to the developer team.
- There might be small changes required in separate parts but not a major change.
- As it requires time, so there must be some time left for the market constraints.
- Risk is high and continuous targets to achieve and report to customer repeatedly.
- It is used when working on a technology is new and requires time to learn.

Advantages Evolutionary Model

- In evolutionary model, a user gets a chance to experiment partially developed system.
- It reduces the error because the core modules get tested thoroughly.

Disadvantages Evolutionary Model

- 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.

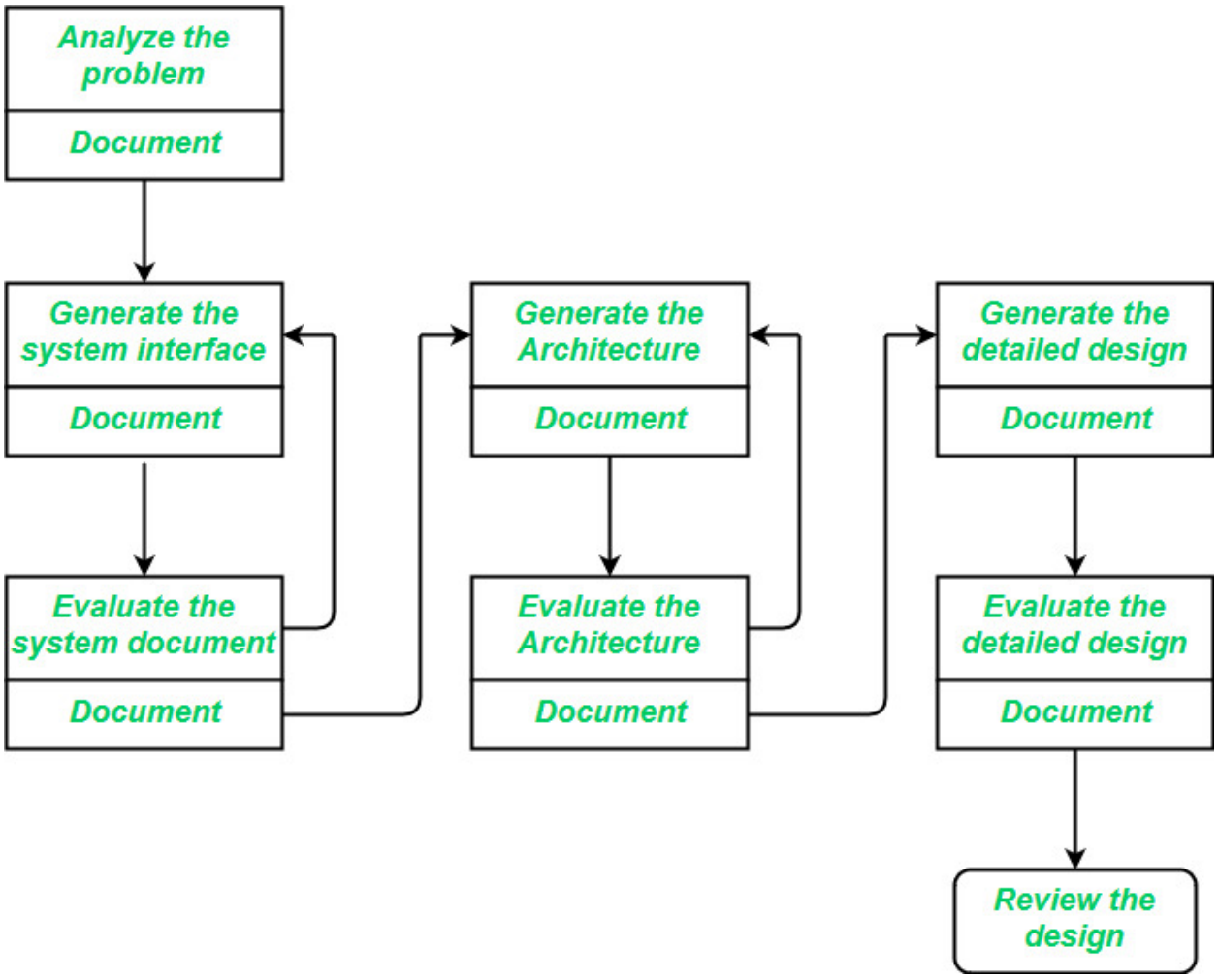
Design process –

The design phase of software development deals with transforming the customer requirements as described in the SRS documents into a form implementable using a programming language. The software design process can be divided into the following three levels or phases of design:

1. Interface Design
2. Architectural Design
3. Detailed Design

Elements of a System

1. **Architecture:** This is the conceptual model that defines the structure, behavior, and views of a system. We can use flowcharts to represent and illustrate the architecture.
2. **Modules:** These are components that handle one specific task in a system. A combination of the modules makes up the system.
3. **Components:** This provides a particular function or group of related functions. They are made up of modules.
4. **Interfaces:** This is the shared boundary across which the components of a system exchange information and relate.
5. **Data:** This is the management of the information and data flow.



Interface Design

Interface design is the specification of the interaction between a system and its environment. This phase proceeds at a high level of abstraction with respect to the inner workings of the system i.e, during interface design, the internal of the systems are completely ignored, and the system is treated as a black box. Attention is focused on the dialogue between the target system and the users, devices, and other systems with which it interacts. The design problem statement produced during the problem analysis step should identify the people, other systems, and devices which are collectively called agents.

Interface design should include the following details:

1. Precise description of events in the environment, or messages from agents to which the system must respond.
2. Precise description of the events or messages that the system must produce.
3. Specification of the data, and the formats of the data coming into and going out of the system.
4. Specification of the ordering and timing relationships between incoming events or messages, and outgoing events or outputs.

Architectural Design

Architectural design is the specification of the major components of a system, their responsibilities, properties, interfaces, and the relationships and interactions between them. In architectural design, the overall structure of the system is chosen, but the internal details of major components are ignored. Issues in architectural design includes:

1. Gross decomposition of the systems into major components.
2. Allocation of functional responsibilities to components.
3. Component Interfaces.
4. Component scaling and performance properties, resource consumption properties, reliability properties, and so forth.
5. Communication and interaction between components.

The architectural design adds important details ignored during the interface design. Design of the internals of the major components is ignored until the last phase of the design.

Detailed Design

Design is the specification of the internal elements of all major system components, their properties, relationships, processing, and often their algorithms and the data structures. The detailed design may include:

1. Decomposition of major system components into program units.
2. Allocation of functional responsibilities to units.
3. User interfaces.
4. Unit states and state changes.
5. Data and control interaction between units.
6. Data packaging and implementation, including issues of scope and visibility of program elements.
7. Algorithms and data structures.

Software design quality –

First of all, the code that we develop is a long-lasting, “living entity”. Once it is created, it evolves over time. In every part of its life cycle, the code is read by many developers many times. Especially if you are developing open-source software or developing software in a big organization. Therefore, it should be easily understandable by the other developers as well.

“Any code of your own that you haven’t looked at for six or more months might as well have been written by someone else.”

Eagleson’s Law

According to studies, developers spend 80% of development time reading the code! So, code is read much more often than it is written.

Code changes are due to several reasons like bug fixing, upgrading the tech stack, and enhancements. Most companies have to spend up to 80% of the development budget for maintaining the code.

The design is good and, if we want to change it, we don’t have to spend too much time and effort. And, also, we should extend it without fear. A good design should be adaptable to the changes,

and the cost of the change should be as minimum as possible. Otherwise, adding new features costs more and more! If the code is maintainable and extendable, you can modify it without fear, and adding new features does not cost too much.

"Maintenance typically claims 40–80% of all project cost."

Barry Boehm

How can we define good design or bad design? It is hard to formalize what good design is, but we all somehow can feel when we see a good design or a bad design. It is one of the things that make software both an art and a science.

Let's talk about the difference between good and bad design with a city analogy. When you visit a city, you walk around to discover the neighborhood. In an organized, clean, and well-maintained one you enjoy and feel the atmosphere. It is the total opposite in a city like in the second picture. These are also valid for software design. To achieve a good design, we should thoroughly think about our design.

Good designs may take longer at the beginning, they but make future changes easier. On the other hand, bad designs may make you have a feature available sooner, but make future changes harder short-term, and long-term.

Of course, the reason for a failed project is not only poor quality. Another fact about quality is the cost of design and the cost of fixing design defects. Studies show that only 10% to 15% of the total cost is designed itself. On the other hand, fixing design defects can cost up to eighty percent of total development.

Like in city architectures in the software, there are also common practices to guide us called design principles and design patterns such as SOLID, YAGNI (You aren't going to need it), DRY(Don't repeat yourself), and don't repeat the effort. They create common solutions to common problems and create a good development atmosphere for all of us once they are applied correctly.

On the other hand, there are also some anti-patterns often called "code smells'' or "bad smells". They are disharmonies and anti-patterns in computer programming code, referring to any symptom in the source code of a program that may contain an area of concern. They can be used as an indication of a weakness in the design. A code smell is a surface indication that usually corresponds to a deeper problem in the system. More specifically, a code smell has to be sniffable; something that's quick to spot. However, this doesn't always indicate a problem.

What we don't want to see in the code is the indicator of the weakness in design that may be slowing down the development of increasing the risk.

"A code smell is a surface indication that usually corresponds to a deeper problem in the system".

Martin Fowler

- ***Functionality***
- ***Performance***
- ***Reliability***
- ***Usability***
- ***Coupling***
- ***Cohesion***

- ***Complexity***
- ***Design structure***
- ***Testability***
- ***Reusability***
- ***…..***

We can imagine external and internal attributes as an iceberg. Users see external attributes like functionality performance reliability, but those are some kind of reflection of internal attributes like coupling, cohesion complexity, and reusability. Some example pictures may help us to figure out how the design quality may affect maintenance cost. Maintenance costs can increase dramatically for some bad designs depending on whatever programming languages you use.
Technical debt, also known as design debt or code debt, is a concept in software development that reflects the implied cost of additional rework caused by choosing an easy solution now instead of using a better approach that would take longer. Ward Cunningham explains Technical Debt with a financial analogy, like financial debt, the technical debt incurs in payment. It is acceptable and manageable until some level if you can control it while maintaining the project schedule and budget. However, *"prevention is better than cure"*, just like for health, this also applies to software, the cost of a defect rises significantly the later in the process it is discovered.

Software Design Concepts

Every software process is characterized by basic concepts along with certain practices or methods. Methods represent the manner through which the concepts are applied. As new technology replaces older technology, many changes occur in the methods that are used to apply the concepts for the development of software. However, the fundamental concepts underlining the software design process remain the same, some of which are described here.

Abstraction

Abstraction refers to a powerful design tool, which allows software designers to consider components at an abstract level, while neglecting the implementation details of the components. **IEEE** defines abstraction as 'a view of a problem that extracts the essential information relevant to a particular purpose and ignores the remainder of the information.' The concept of abstraction can be used in two ways: as a process and as an entity. As a **process,** it refers to a mechanism of hiding irrelevant details and representing only the essential features of an item so that one can focus on important things at a time. As an **entity,** it refers to a model or view of an item.

Each step in the software process is accomplished through various levels of abstraction. At the highest level, an outline of the solution to the problem is presented whereas at the lower levels, the solution to the problem is presented in detail. For example, in the requirements analysis phase, a solution to the problem is presented using the language of problem environment and as we proceed through the software process, the abstraction level reduces and at the lowest level, source code of the software is produced.

There are three commonly used abstraction mechanisms in software design, namely, functional abstraction, data abstraction and control abstraction. All these mechanisms allow us to control the complexity of the design process by proceeding from the abstract design model to concrete design model in a systematic manner.

- **Functional abstraction:** This involves the use of parameterized subprograms. Functional abstraction can be generalized as collections of subprograms referred to as 'groups'. Within these groups there exist routines which may be visible or hidden. Visible routines can be used within the containing groups as well as within other groups, whereas hidden routines are hidden from other groups and can be used within the containing group only.

- **Data abstraction:** This involves specifying data that describes a data object. For example, the data object *window* encompasses a set of attributes (window type, window dimension) that describe the window object clearly. In this abstraction mechanism, representation and manipulation details are ignored.

- **Control abstraction:** This states the desired effect, without stating the exact mechanism of control. For example, if and while statements in programming languages (like C and C++) are abstractions of machine code implementations, which involve conditional instructions. In the architectural design level, this abstraction mechanism permits specifications of sequential subprogram and exception handlers without the concern for exact details of implementation.

Architecture

Software architecture refers to the structure of the system, which is composed of various components of a program/ system, the attributes (properties) of those components and the relationship amongst them. The software architecture enables the software engineers to analyze the software design efficiently. In addition, it also helps them in decision-making and handling risks. The software architecture does the following.

- Provides an insight to all the interested stakeholders that enable them to communicate with each other
- Highlights early design decisions, which have great impact on the software engineering activities (like coding and testing) that follow the design phase
- Creates intellectual models of how the system is organized into components and how these components interact with each other.

Currently, software architecture is represented in an informal and unplanned manner. Though the architectural concepts are often represented in the infrastructure (for supporting particular architectural styles) and the initial stages of a system configuration, the lack of an explicit independent characterization of architecture restricts the advantages of this design concept in the present scenario.

Note that software architecture comprises two elements of design model, namely, data design and architectural design.

Patterns

A pattern provides a description of the solution to a recurring design problem of some specific domain in such a way that the solution can be used again and again. The objective of each pattern is to provide an insight to a designer who can determine the following.

- Whether the pattern can be reused
- Whether the pattern is applicable to the current project
- Whether the pattern can be used to develop a similar but functionally or structurally different design pattern.

Types of Design Patterns

Software engineer can use the design pattern during the entire software design process. When the analysis model is developed, the designer can examine the problem description at different levels of abstraction to determine whether it complies with one or more of the following types of design patterns.

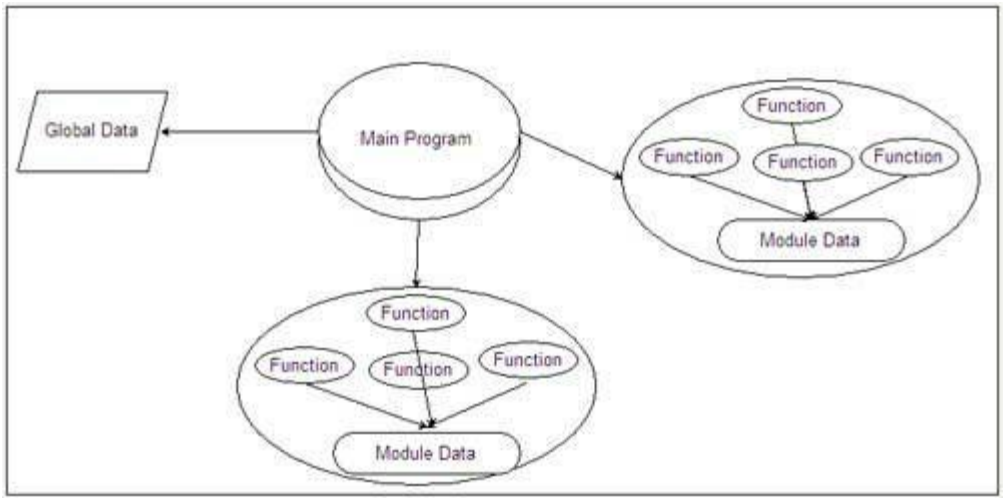
- **Architectural patterns:** These patterns are high-level strategies that refer to the overall structure and organization of a software system. That is, they define the elements of a software system such as subsystems, components, classes, etc. **In** addition, they also indicate the relationship between the elements along with the rules and guidelines for specifying these relationships. Note that architectural patterns are often considered equivalent to software architecture.

- **Design patterns:** These patterns are medium-level strategies that are used to solve design problems. They provide a means for the refinement of the elements (as defined by architectural pattern) of a software system or the relationship among them. Specific design elements such as relationship among components or mechanisms that affect component-to-component interaction are addressed by design patterns. Note that design patterns are often considered equivalent to software components.

- **Idioms:** These patterns are low-level patterns, which are programming-language specific. They describe the implementation of a software component, the method used for interaction among software components, etc., in a specific programming language. Note that idioms are often termed as coding patterns.

Modularity

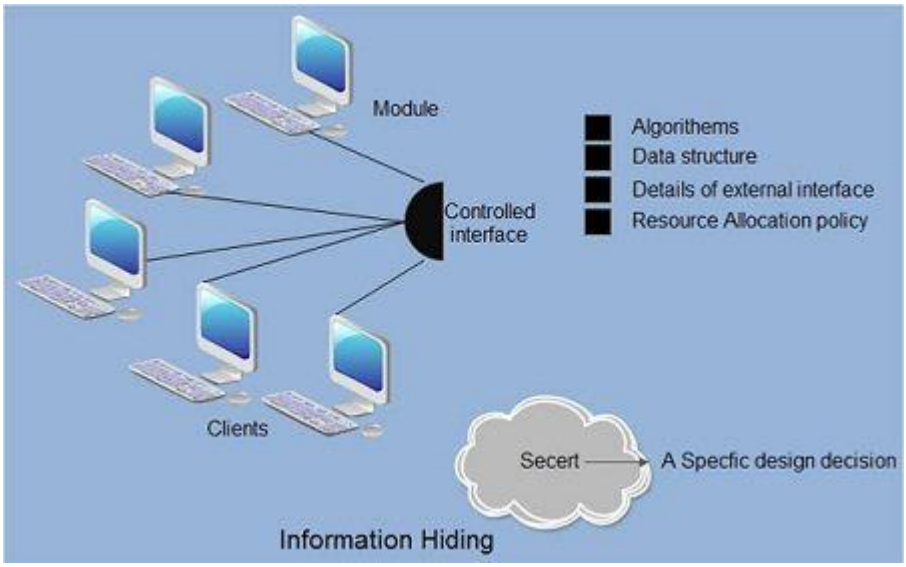
Modularity is achieved by dividing the software into uniquely named and addressable components, which are also known as **modules.** A complex system (large program) is partitioned into a set of discrete modules in such a way that each module can be developed independent of other modules. After developing the modules, they are integrated together to meet the software requirements. Note that larger the number of modules a system is divided into, greater will be the effort required to integrate the modules.



Modularizing a design helps to plan the development in a more effective manner, accommodate changes easily, conduct testing and debugging effectively and efficiently, and conduct maintenance work without adversely affecting the functioning of the software.

Information Hiding

Modules should be specified and designed in such a way that the data structures and processing details of one module are not accessible to other modules. They pass only that much information to each other, which is required to accomplish the software functions. The way of hiding unnecessary details is referred to as **information hiding. IEEE** defines information hiding as 'the technique of encapsulating software design decisions in modules in such a way that the module's interfaces reveal as little as possible about the module's inner workings; thus each module is a 'black box' to the other modules in the system.



Information hiding is of immense use when modifications are required during the testing and maintenance phase. Some of the advantages associated with information hiding are listed below.

- Leads to low coupling

- Emphasizes communication through controlled interfaces
- Decreases the probability of adverse effects
- Restricts the effects of changes in one component on others
- Results in higher quality software.

Stepwise Refinement

Stepwise refinement is a top-down design strategy used for decomposing a system from a high level of abstraction into a more detailed level (lower level) of abstraction. At the highest level of abstraction, function or information is defined conceptually without providing any information about the internal workings of the function or internal structure of the data. As we proceed towards the lower levels of abstraction, more and more details are available.

Software designers start the stepwise refinement process by creating a sequence of compositions for the system being designed. Each composition is more detailed than the previous one and contains more components and interactions. The earlier compositions represent the significant interactions within the system, while the later compositions show in detail how these interactions are achieved.

To have a clear understanding of the concept, let us consider an example of stepwise refinement. Every computer program comprises input, process, and output.

- INPUT

- Get user's name (string) through a prompt.
- Get user's grade (integer from 0 to 100) through a prompt and validate.

- PROCESS
- OUTPUT

This is the first step in refinement. The input phase can be refined further as given here.

- INPUT
  - Get user's name through a prompt.
  - Get user's grade through a prompt.
  - While (invalid grade)

Refactoring

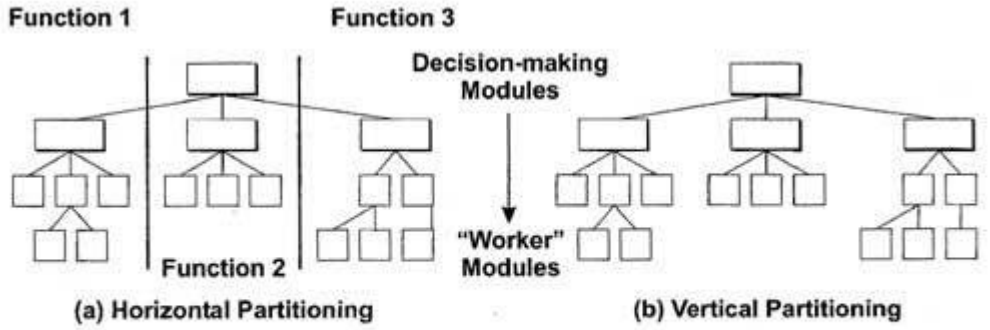
Refactoring is an important design activity that reduces the complexity of module design keeping its behaviour or function unchanged. Refactoring can be defined as a process of modifying a software system to improve the internal structure of design without changing its external behavior. During the refactoring process, the existing design is checked for any type of flaws like redundancy, poorly constructed algorithms and data structures, etc., in order to improve the design. For example, a design model might yield a component which exhibits low cohesion (like a component performs four functions that have a limited relationship with one another). Software designers may decide to refactor the component into four different components, each exhibiting high cohesion. This leads to easier integration, testing, and maintenance of the software components.

Structural Partitioning

When the architectural style of a design follows a hierarchical nature, the structure of the program can be partitioned either horizontally or vertically. In **horizontal partitioning,** the control modules are used to communicate between functions and execute the functions. Structural partitioning provides the following benefits.

- The testing and maintenance of software becomes easier.
- The negative impacts spread slowly.
- The software can be extended easily.

Besides these advantages, horizontal partitioning has some disadvantage also. It requires to pass more data across the module interface, which makes the control flow of the problem more complex. This usually happens in cases where data moves rapidly from one function to another.



Horizontal and Vertical Partitioning

In **vertical partitioning**, the functionality is distributed among the modules–in a top-down manner. The modules at the top level called **control modules** perform the decision-making and do little processing whereas the modules at the low level called **worker modules** perform all input, computation and output tasks.

Concurrency

Computer has limited resources and they must be utilized efficiently as much as possible. To utilize these resources efficiently, multiple tasks must be executed concurrently. This requirement makes concurrency one of the major concepts of software design. Every system must be designed to allow multiple processes to execute concurrently, whenever possible. For example, if the current process is waiting for some event to occur, the system must execute some other process in the mean time.

However, concurrent execution of multiple processes sometimes may result in undesirable situations such as an inconsistent state, deadlock, etc. For example, consider two processes A and B and a data item Q1 with the value '200'. Further, suppose A and B are being executed concurrently and firstly A reads the value of Q1 (which is '200') to add '100' to it. However, before A updates es the value of Q1, B reads the value ofQ1 (which is still '200') to add '50' to it. In this situation, whether A or B first updates the value of Q1, the value of would definitely be

wrong resulting in an inconsistent state of the system. This is because the actions of A and B are not synchronized with each other. Thus, the system must control the concurrent execution and synchronize the actions of concurrent processes.

One way to achieve synchronization is mutual exclusion, which ensures that two concurrent processes do not interfere with the actions of each other. To ensure this, mutual exclusion may use locking technique. In this technique, the processes need to lock the data item to be read or updated. The data item locked by some process cannot be accessed by other processes until it is unlocked. It implies that the process, that needs to access the data item locked by some other process, has to wait.

Developing a Design Model

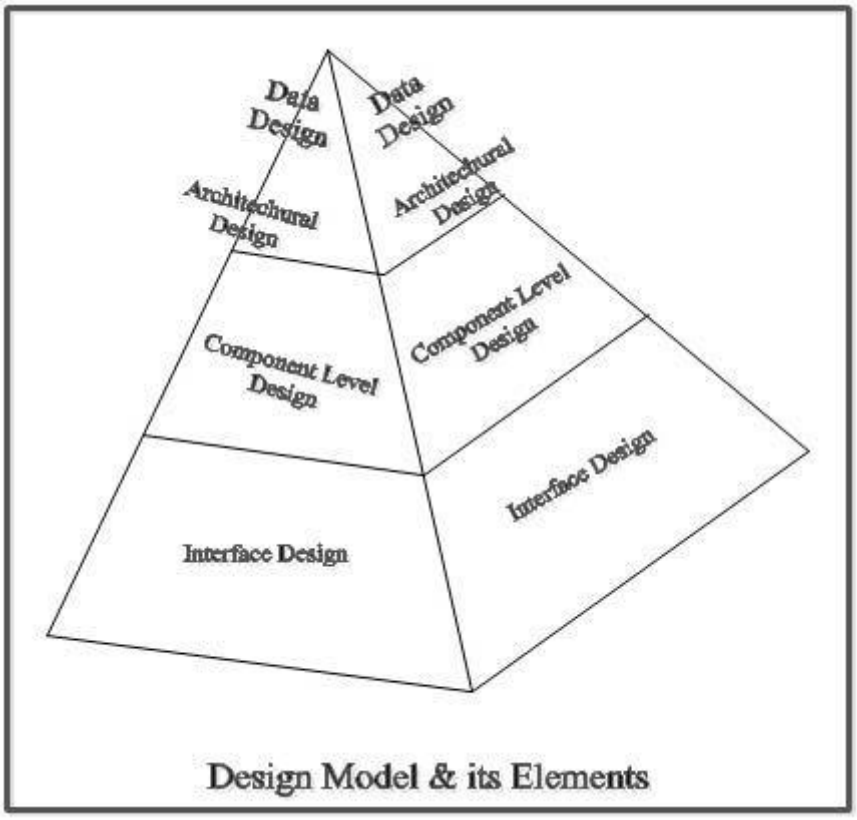
To develop a complete specification of design (design model), four design models are needed. These models are listed below.

- **Data design:** This specifies the data structures for implementing the software by converting data objects and their relationships identified during the analysis phase. Various studies suggest that design engineering should begin with data design, since this design lays the foundation for all other design models.

- **Architectural design:** This specifies the relationship between the structural elements of the software, design patterns, architectural styles, and the factors affecting the ways in which architecture can be implemented.

- **Component-level design:** This provides the detailed description of how structural elements of software will actually be implemented.

- **Interface design:** This depicts how the software communicates with the system that interoperates with it and with the end-users.



Design Model & its Elements

Software Architecture –

Software Architecture defines fundamental organization of a system and more simply defines a structured solution. It defines how components of a software system are assembled, their relationship and communication between them. It serves as a blueprint for software application and development basis for developer team.

Software architecture defines a list of things which results in making many things easier in the software development process.

- A software architecture defines structure of a system.
- A software architecture defines behavior of a system.
- A software architecture defines component relationship.
- A software architecture defines communication structure.
- A software architecture balances stakeholder's needs.
- A software architecture influences team structure.
- A software architecture focuses on significant elements.
- A software architecture captures early design decisions.

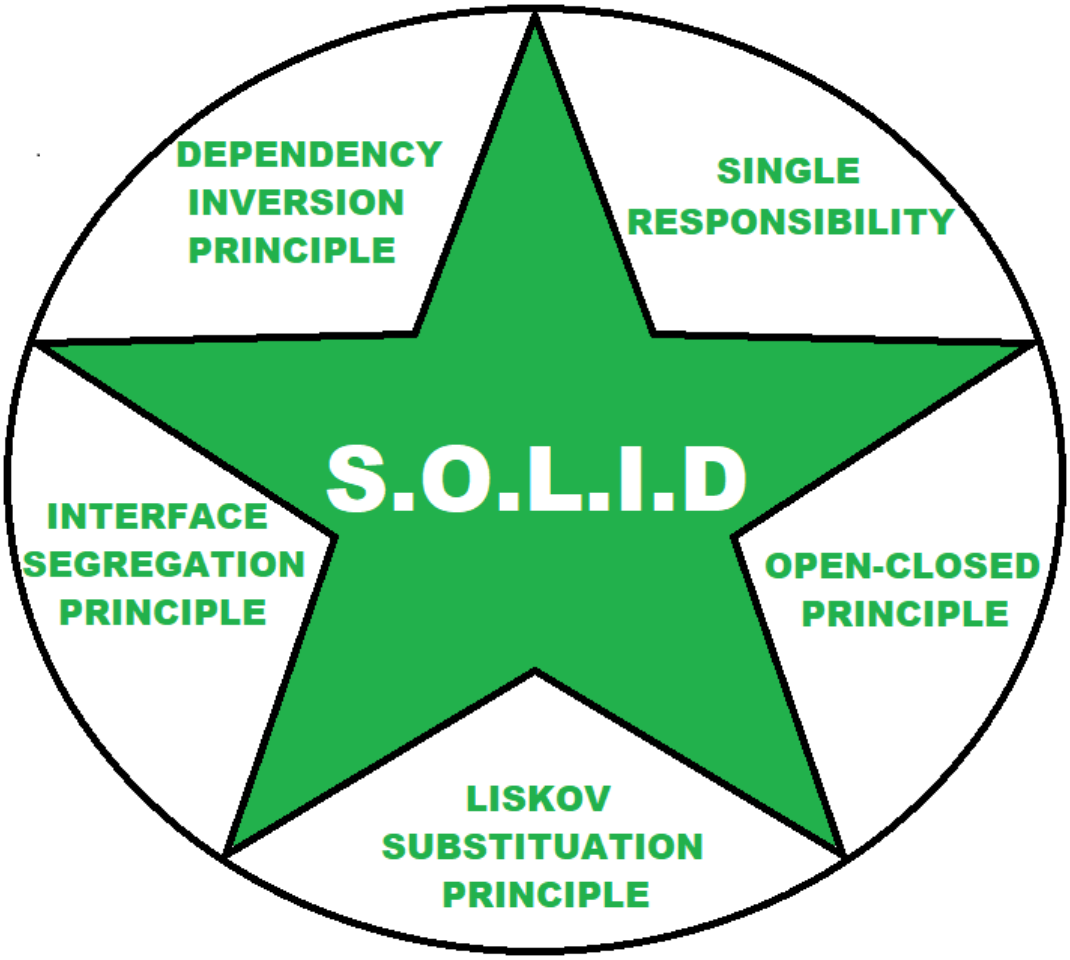
Characteristics of Software Architecture :

Architects separate architecture characteristics into broad categories depending upon operation, rarely appearing requirements, structure etc. Below some important characteristics which are commonly considered are explained.

- **Operational Architecture Characteristics :**
  1. Availability
  2. Performance
  3. Reliability
  4. Low fault tolerance
  5. Scalability
- **Structural Architecture Characteristics :**
  1. Configurability
  2. Extensibility
  3. Supportability
  4. Portability
  5. Maintainability
- **Cross-Cutting Architecture Characteristics :**
  1. Accessibility
  2. Security
  3. Usability
  4. Privacy
  5. Feasibility

SOLID principles of Software architecture :

Each character of the word *SOLID* defines one principle of software architecture. This SOLID principle is followed to avoid product strategy mistakes. A software architecture must adhere to SOLID principle to avoid any architectural or developmental failure.



1. **Single Responsibility –**
   Each services should have a single objective.
2. **Open-Closed Principle –**
   Software modules should be independent and expandable.
3. **Liskov Substitution Principle –**
   Independent services should be able to communicate and substitute each other.
4. **Interface Segregation Principle –**
   Software should be divided into such microservices there should not be any redundancies.
5. **Dependency Inversion Principle –**
   Higher-levels modules should not be depending on low-lower-level modules and changes in higher level will not affect to lower level.

Importance of Software Architecture :

Software architecture comes under design phase of software development life cycle. It is one of initial step of whole software development process. Without software architecture

proceeding to software development is like building a house without designing architecture of house.

So software architecture is one of important part of software application development. In technical and developmental aspects point of view below are reasons software architecture are important.

- Selects quality attributes to be optimized for a system.
- Facilitates early prototyping.
- Allows to be built a system in component wise.
- Helps in managing the changes in System.

Besides all these **software architecture** is also important for many other factors like quality of software, reliability of software, maintainability of software, Supportability of software and performance of software and so on.

Advantages of Software Architecture :

- Provides a solid foundation for software project.
- Helps in providing increased performance.
- Reduces development cost.

Disadvantages of Software Architecture :

- Sometimes getting good tools and standardization becomes a problem for software architecture.
- Initial prediction of success of project based on architecture is not always possible.

From above it's clear how much important a **software architecture** for the development of a software application. So a good software architecture is also responsible for delivering a good quality software product.

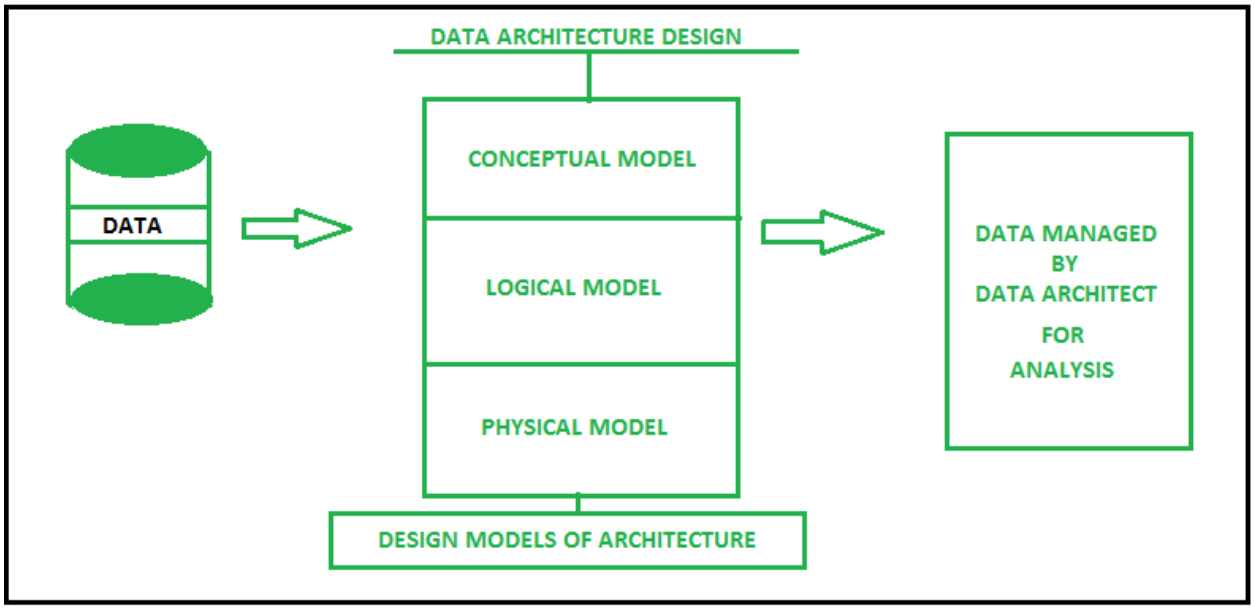
Data architecture Design :

Data architecture design is set of standards which are composed of certain policies, rules, models and standards which manages, what type of data is collected, from where it is collected, the arrangement of collected data, storing that data, utilizing and securing the data into the systems and data warehouses for further analysis.

Data is one of the essential pillars of enterprise architecture through which it succeeds in the execution of business strategy.

Data architecture design is important for creating a vision of interactions occurring between data systems, like for example if data architect wants to implement data integration, so it will need interaction between two systems and by using data architecture the visionary model of data interaction during the process can be achieved.

Data architecture also describes the type of data structures applied to manage data and it provides an easy way for data preprocessing. The data architecture is formed by dividing into three essential models and then are combined :



- **Conceptual model –**
  It is a business model which uses Entity Relationship (ER) model for relation between entities and their attributes.
- **Logical model –**
  It is a model where problems are represented in the form of logic such as rows and column of data, classes, xml tags and other DBMS techniques.
- **Physical model –**
  Physical models holds the database design like which type of database technology will be suitable for architecture.

A data architect is responsible for all the design, creation, manage, deployment of data architecture and defines how data is to be stored and retrieved, other decisions are made by internal bodies.

Factors that influence Data Architecture :
Few influences that can have an effect on data architecture are business policies, business requirements, Technology used, economics, and data processing needs.

- **Business requirements –**
  These include factors such as the expansion of business, the performance of the system access, data management, transaction management, making use of raw data by converting them into image files and records, and then storing in data warehouses. Data warehouses are the main aspects of storing transactions in business.
- **Business policies –**
  The policies are rules that are useful for describing the way of processing data. These policies are made by internal organizational bodies and other government agencies.
- **Technology in use –**
  This includes using the example of previously completed data architecture design and also using existing licensed software purchases, database technology.

- **Business economics –**
  The economical factors such as business growth and loss, interest rates, loans, condition of the market, and the overall cost will also have an effect on design architecture.
- **Data processing needs –**
  These include factors such as mining of the data, large continuous transactions, database management, and other data preprocessing needs.

Different Software Architecture Patterns :

1. Layered Pattern
2. Client-Server Pattern
3. Event-Driven Pattern
4. Microkernel Pattern
5. Microservices Pattern

Let's see one by one in detail.

1. Layered Pattern :

As the name suggests, components(code) in this pattern are separated into layers of subtasks and they are arranged one above another.
Each layer has unique tasks to do and all the layers are independent of one another. Since each layer is independent, one can modify the code inside a layer without affecting others.

It is the most commonly used pattern for designing the majority of software. This layer is also known as 'N-tier architecture'. Basically, this pattern has 4 layers.

1. Presentation layer (The user interface layer where we see and enter data into an application.)
2. Business layer (this layer is responsible for executing business logic as per the request.)
3. Application layer (this layer acts as a medium for communication between the 'presentation layer' and 'data layer'.
4. Data layer (this layer has a database for managing data.)

2. Client-Server Pattern :

The client-server pattern has two major entities. They are a server and multiple clients.
Here the server has resources(data, files or services) and a client requests the server for a particular resource. Then the server processes the request and responds back accordingly.

Examples of software developed in this pattern:

- Email.
- WWW.
- File sharing apps.
- Banking, etc…

So this pattern is suitable for developing the kind of software listed in the examples.

3. Event-Driven Pattern :

Event-Driven Architecture is an agile approach in which services (operations) of the software are triggered by events.
Well, what does an event mean?

When a user takes action in the application built using the EDA approach, a state change happens and a reaction is generated that is called an event.

Eg: A new user fills the signup form and clicks the signup button on Facebook and then a FB account is created for him, which is an event.
Ideal for:

Building websites with JavaScript and e-commerce websites in general.

4. Microkernel Pattern :

Microkernel pattern has two major components. They are a core system and plug-in modules.

- The core system handles the fundamental and minimal operations of the application.
- The plug-in modules handle the extended functionalities (like extra features) and customized processing.

Let's imagine, you have successfully built a chat application. And the basic functionality of the app is that you can text with people across the world without an internet connection. After some time, you would like to add a voice messaging feature to the application, then you are adding the feature successfully. You can add that feature to the already developed application because the microkernel pattern facilitates you to add features as plug-ins. Microkernel pattern is ideal for: Product-based applications and scheduling applications. We love new features that keep giving dopamine boost to our brain. Such as Instagram reels, YouTube Shorts and a lot more that feasts us digitally. So this pattern is mostly preferred for app development.

5. Microservices Pattern :

The collection of small services that are combined to form the actual application is the concept of microservices pattern. Instead of building a bigger application, small programs are built for every service (function) of an application independently. And those small programs are bundled together to be a full-fledged application.

So adding new features and modifying existing microservices without affecting other microservices are no longer a challenge when an application is built in a microservices pattern.

Modules in the application of microservices patterns are loosely coupled. So they are easily understandable, modifiable and scalable.

Example Netflix is one of the most popular examples of software built-in microservices architecture. This pattern is most suitable for websites and web apps having small components.

Principles of software testing – Software Testing

Software testing is the process of executing a program to find the error. To make our software perform well it should be error-free. If testing is done successfully it will remove all the errors from the software. Software Testing is a process of evaluation of the software on the requirements gathered from system specifications and users. Software Testing is done at every phase level in software development life cycle(SDLC). Can also be done at a module level in the program code. Validation and Verification are two main things that are included in Software testing. also, Software Testing is very important otherwise the software bugs can be dangerous.

Principles of software testing:

- Testing shows the presence of defects
- Exhaustive testing is not possible
- Early testing
- Defect clustering
- Pesticide paradox
- Testing is context-dependent
- Absence of errors fallacy
- Testing Is a Risk-Based Activity
- Testing Cannot Prove Software Correctness
- Absence-of-Error Fallacy
- Automated Testing Complements Manual Testing
- Testing Is Iterative

Testing shows the presence of defects: The goal of software testing is to make the software fail. Software testing reduces the presence of defects. Software testing talks about the presence of defects and doesn't talk about the absence of defects. Software testing can ensure that defects are present but it can not prove that software is defect-free. Even multiple testing can never ensure that software is 100% bug-free. Testing can reduce the number of defects but not remove all defects.

Exhaustive testing is not possible: It is the process of testing the functionality of the software in all possible inputs (valid or invalid) and pre-conditions is known as exhaustive testing. Exhaustive testing is impossible means the software can never test at every test case. It can test only some test cases and assume that the software is correct and it will produce the correct output in every test case. If the software will test every test case then it will take more cost, effort, etc., which is impractical.

Early Testing: To find the defect in the software, early test activity shall be started. The defect detected in the early phases of SDLC will be very less expensive. For better performance of software, software testing will start at the initial phase i.e. testing will perform at the requirement analysis phase.

Defect clustering: In a project, a small number of modules can contain most of the defects. The Pareto Principle for software testing states that 80% of software defects come from 20% of modules.

Pesticide paradox: Repeating the same test cases, again and again, will not find new bugs. So it is necessary to review the test cases and add or update test cases to find new bugs.

Testing is context-dependent: The testing approach depends on the context of the software developed. Different types of software need to perform different types of testing. For example, The testing of the e-commerce site is different from the testing of the Android application.

Absence of errors fallacy: If a built software is 99% bug-free but does not follow the user requirement then it is unusable. It is not only necessary that software is 99% bug-free but it is also mandatory to fulfill all the customer requirements.

Testing Is a Risk-Based Activity: Testing efforts should be prioritized according to the risks associated with different features or functionalities. Focus testing on areas that are critical to the success of the application.

Testing Cannot Prove Software Correctness: Testing demonstrates the presence of defects but cannot prove that the software is entirely correct. It provides confidence in the software's behavior within the scope of the tests performed.

Absence-of-Error Fallacy: The absence of detected defects does not proof that the software is error-free. It implies that the testing process has not found any issues within the defined scope and constraints.
Automated Testing Complements Manual Testing: Combining Both automated and manual testing allows for more comprehensive test coverage.
Testing Is Iterative: Testing is an iterative process that continues throughout the software development life cycle.
Types of Software Testing:

1. Unit Testing
2. Integration Testing
3. Regression Testing
4. Smoke Testing
5. System Testing
6. Alpha Testing
7. Beta Testing
8. Performance Testing

1. Unit Testing
Unit tests are typically written by developers as they write the code for a given unit. They are usually written in the same programming language as the software and use a testing framework or library that provides the necessary tools for creating and running the tests. These frameworks often include assertion libraries, which allow developers to write test cases that check the output of a given unit against expected results. The tests are usually run automatically and continuously as part of the software build process, and the results are typically displayed in a test runner or a continuous integration tool.
Unit testing has several benefits, including:
Early detection and isolation of defects, which can save time and money by allowing developers to fix errors before they become more costly to fix.
Improved software quality and maintainability, as unit tests help to ensure that code changes do not break existing functionality.
Increased developer confidence, as developers can make changes to the code with the knowledge that any errors will be caught by the unit tests.
Facilitation of test-driven development, a software development methodology in which tests are written before code is written, ensuring that code is written to meet the requirements.
Overall, Unit testing is an essential part of software development that helps to ensure the quality and reliability of the software, by identifying errors early on in the development process.

2. Integration Testing
Integration testing is a software testing method in which individual units or components of a software application are combined and tested as a group. The goal of integration testing is to validate that the interactions between the units or components of the software work as expected and that the software as a whole functions correctly.
Integration testing is typically performed after unit testing and before system testing. It is usually done by developers and test engineers, and it is usually carried out at the module level. Integration tests are typically automated and run frequently, as part of the software build process, to ensure that the software remains stable and free of defects over time.
Integration testing is done to verify that different components or modules of the software work together as expected, and to identify and fix any issues that might arise due to interactions between the modules. These tests can include testing different combinations of inputs, testing how the software handles different types of data, and testing how the software handles different error conditions.
Integration testing has several benefits, including:
Detection of defects that may not be discovered during unit testing, as it examines the interactions between components
Improved system design, as integration testing can help identify design weaknesses
Improved software quality and reliability, as integration testing helps to ensure that the software as a whole

functions correctly.
Facilitation of continuous integration and delivery, as integration testing helps to ensure that changes to the software do not break existing functionality
Overall, integration testing is an essential part of software development that helps to ensure the quality and reliability of the software by identifying defects in the interactions between the units and components of the software early on in the development process.

3. Regression Testing

Regression testing is a software testing method in which previously developed and tested software is retested after it has been modified or changed. The goal of regression testing is to ensure that any changes to the software have not introduced new bugs or broken existing functionality. It is typically done to verify that changes such as bug fixes, new features, or updates to existing features have not affected the overall functionality of the software.
Regression testing is typically performed after unit testing and integration testing. It is usually done by developers and test engineers and it is usually carried out by re-running a suite of previously passed test cases. The test cases are chosen to cover the areas of the software that were affected by the changes, and to ensure that the most critical functionality of the software is still working correctly. Regression testing is typically automated and run frequently, as part of the software build process, to ensure that the software remains stable and free of defects over time.
Regression testing has several benefits, including:
Early detection and isolation of defects, which can save time and money by allowing developers to fix errors before they become more costly to fix.
Improved software quality and maintainability, as regression testing helps to ensure that code changes do not break existing functionality
Increased developer and user confidence, as regression testing helps to ensure that the software is still working correctly after changes have been made
Facilitation of continuous integration and delivery, as regression testing helps to ensure that changes to the software can be safely released.
Overall, regression testing is an essential part of software development that helps to ensure

4. Smoke Testing

Smoke testing, also known as “Build Verification Testing” or “Build Acceptance Testing”, is a software testing method in which a minimal set of tests are run on a new build of a software application to determine if it is stable enough to proceed with further testing. The goal of smoke testing is to quickly identify and isolate major issues with the software build, so that development can be halted if the build is found to be too unstable or unreliable.
Smoke testing is typically performed early in the software testing process, after the software has been built and before more extensive testing is done. It is usually done by developers and test engineers and it is usually carried out by running a small set of critical test cases that exercise the most important functionality of the software. Smoke tests are usually automated and can be run as part of the software build process.
Smoke testing has several benefits, including:
Early identification of major issues, which can save time and money by allowing developers to fix errors before they become more costly to fix.
Improved software quality and reliability, as smoke testing helps to ensure that the software is stable enough to proceed with further testing
Facilitation of continuous integration and delivery, as smoke testing helps to ensure that new builds of the software are stable and reliable before they are released.
Overall, smoke testing is an important part of software development that helps to ensure the quality and reliability of the software by identifying major issues early on in the development process. It helps to quickly determine if a new build of the software is stable enough to proceed with further testing, providing increased confidence in the software to the development team and end-users.

5. System Testing

System testing is a software testing method in which an entire software system is tested as a whole, in order to ensure that it meets the requirements and specifications that it was designed for. The goal of system

testing is to validate that the software system behaves as expected when it is used in its intended environment, and that it meets all the requirements for functionality, performance, security, and usability. System testing is typically performed after unit testing, integration testing, and regression testing. It is usually done by test engineers and it is usually carried out by running a set of test cases that cover all the functionality of the software. The test cases are chosen to cover the requirements and specifications of the software, and to ensure that the software behaves correctly under different conditions and scenarios. System testing is typically automated and run frequently, as part of the software build process, to ensure that the software remains stable and free of defects over time.

System testing has several benefits, including:

Early detection and isolation of defects, which can save time and money by allowing developers to fix errors before they become more costly to fix.

Improved software quality and reliability, as system testing helps to ensure that the software meets all the requirements and specifications that it was designed for.

Increased user confidence, as system testing helps to ensure that the software behaves correctly when it is used in its intended environment.

Facilitation of acceptance testing, as system testing helps to ensure that the software is ready for release.

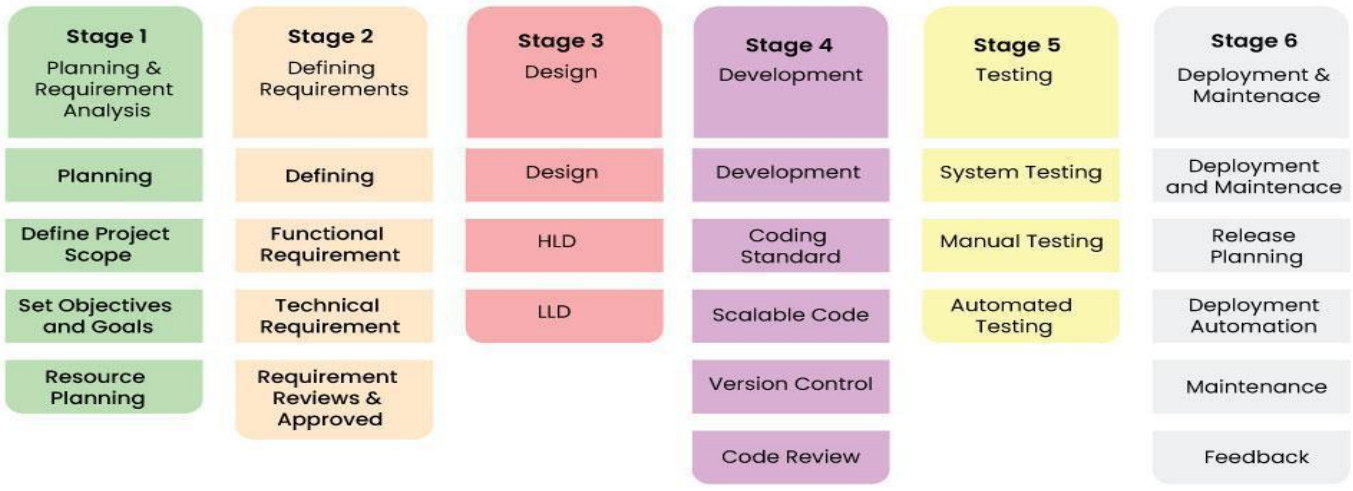
Overall, system testing is an essential part of software development that helps to ensure the quality and reliability of the software by identifying defects early on in the development process. It helps to ensure that the software meets all the requirements and specifications that it was designed for, providing increased confidence in the software to the development team and end-users.

What is Software Development Life Cycle (SDLC)?

SDLC is a process followed for software building within a software organization. SDLC consists of a precise plan that describes how to develop, maintain, replace, and enhance specific software. The life cycle defines a method for improving the quality of software and the all-around development process.

Stages of the Software Development Life Cycle

SDLC specifies the task(s) to be performed at various stages by a software engineer or developer. It ensures that the end product is able to meet the customer's expectations and fits within the overall budget. Hence, it's vital for a software developer to have prior knowledge of this software development process.

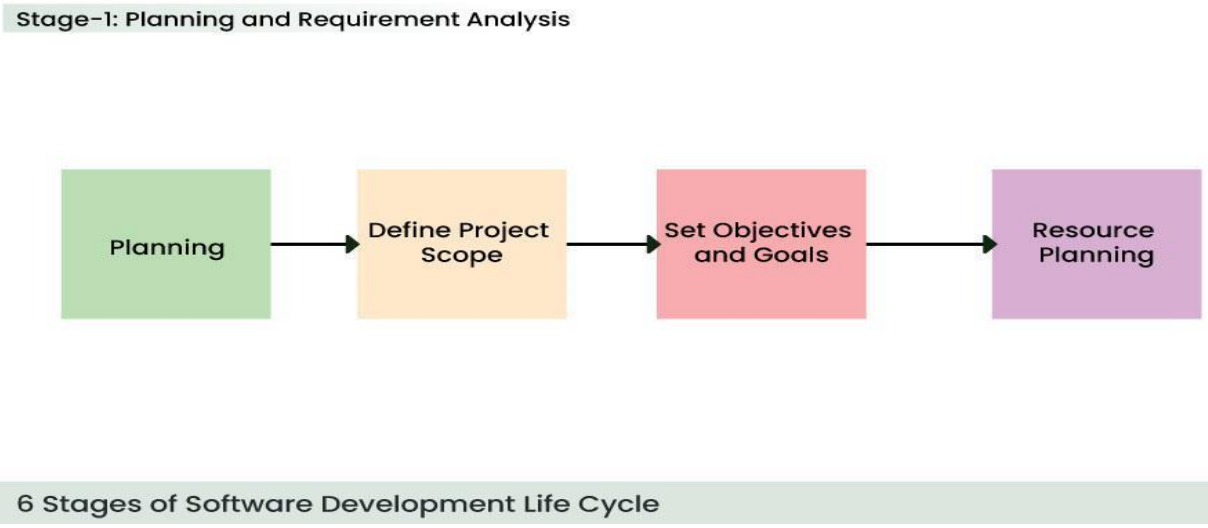


6 Stages of Software Development Life Cycle

Stage-1: Planning and Requirement Analysis

Planning is a crucial step in everything, just as in software development. In this same stage, requirement analysis is also performed by the developers of the organization. This is attained from customer inputs, and sales department/market surveys.

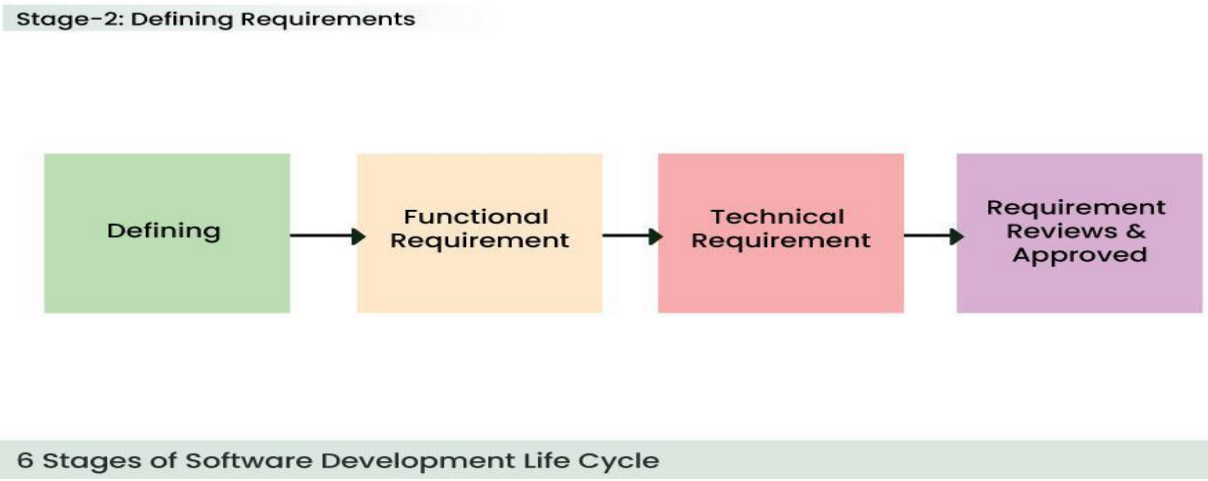
The information from this analysis forms the building blocks of a basic project. The quality of the project is a result of planning. Thus, in this stage, the basic project is designed with all the available information.



Stage-1 : Planning and Requirement Analysis

Stage-2: Defining Requirements

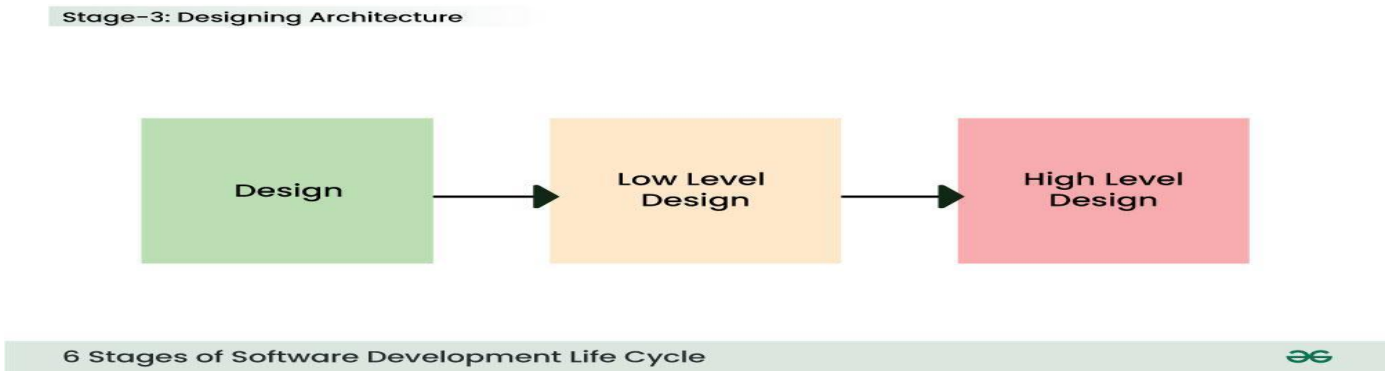
In this stage, all the requirements for the target software are specified. These requirements get approval from customers, market analysts, and stakeholders.
This is fulfilled by utilizing SRS (Software Requirement Specification). This is a sort of document that specifies all those things that need to be defined and created during the entire project cycle.



Stage-2 : Defining Requirements

Stage-3: Designing Architecture

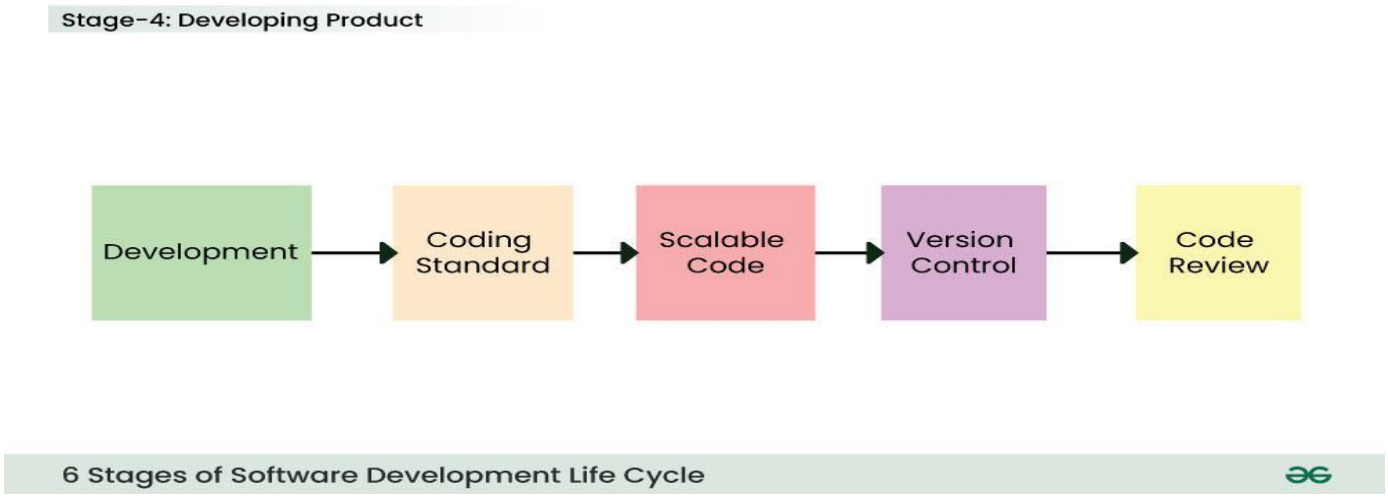
SRS is a reference for software designers to come up with the best architecture for the software. Hence, with the requirements defined in SRS, multiple designs for the product architecture are present in the Design Document Specification (DDS).
This DDS is assessed by market analysts and stakeholders. After evaluating all the possible factors, the most practical and logical design is chosen for development.



Stage 3: Design

Stage-4: Developing Product

At this stage, the fundamental development of the product starts. For this, developers use a specific programming code as per the design in the DDS. Hence, it is important for the coders to follow the protocols set by the association. Conventional programming tools like compilers, interpreters, debuggers, etc. are also put into use at this stage. Some popular languages like C/C++, Python, Java, etc. are put into use as per the software regulations.

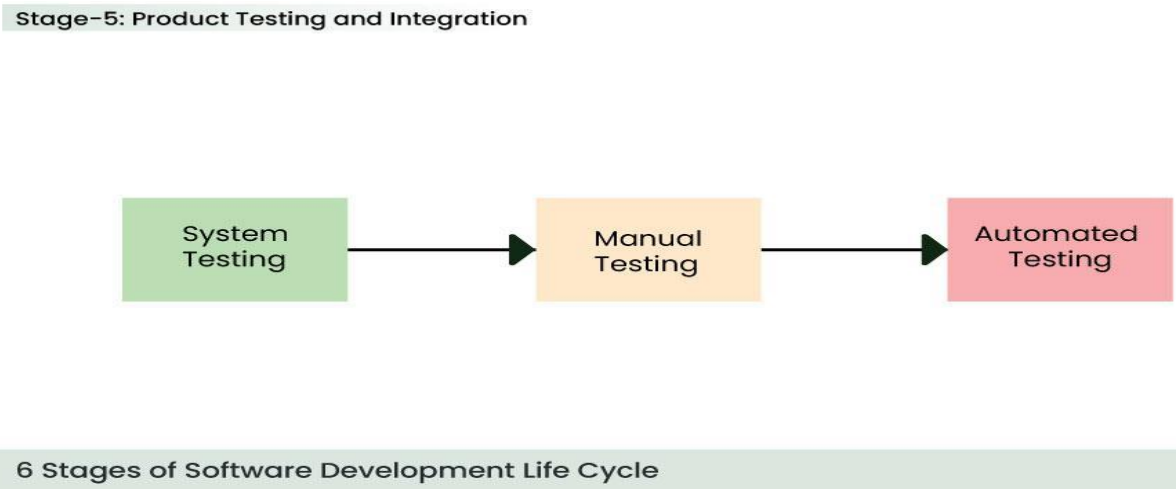


Stage 4: Development

Stage-5: Product Testing and Integration

After the development of the product, testing of the software is necessary to ensure its smooth execution. Although, minimal testing is conducted at every stage of SDLC. Therefore, at this stage, all the probable flaws are tracked, fixed, and retested. This ensures that the product confronts the quality requirements of SRS.

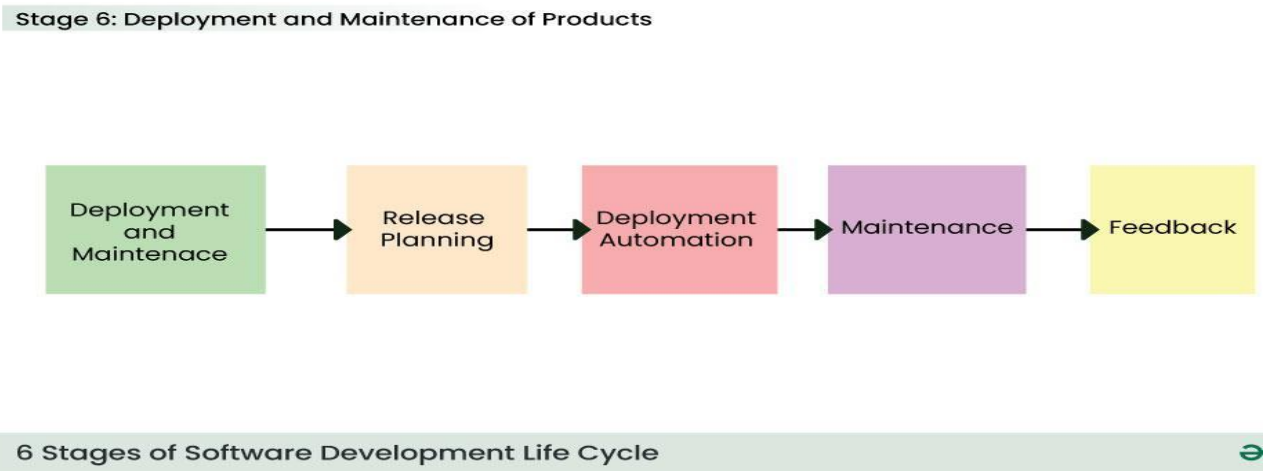
Documentation, Training, and Support: Software documentation is an essential part of the software development life cycle. A well-written document acts as a tool and means to information repository necessary to know about software processes, functions, and maintenance. Documentation also provides information about how to use the product. Training in an attempt to improve the current or future employee performance by increasing an employee's ability to work through learning, usually by changing his attitude and developing his skills and understanding.



Stage 5: Testing

Stage-6: Deployment and Maintenance of Products

After detailed testing, the conclusive product is released in phases as per the organization's strategy. Then it is tested in a real industrial environment. It is important to ensure its smooth performance. If it performs well, the organization sends out the product as a whole. After retrieving beneficial feedback, the company releases it as it is or with auxiliary improvements to make it further helpful for the customers. However, this alone is not enough. Therefore, along with the deployment, the product's supervision.



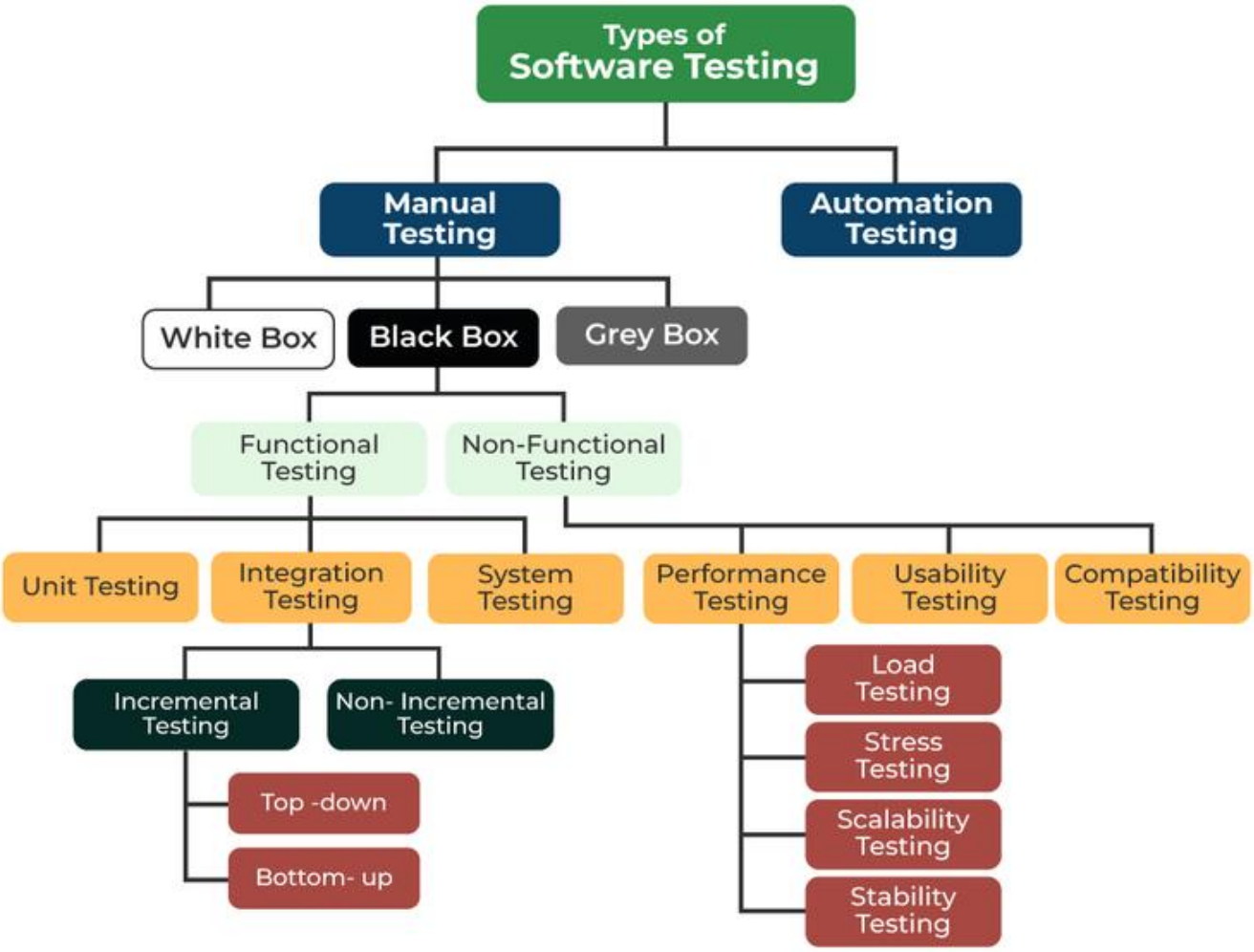
Types of Software Testing

Testing is the process of executing a program to find errors. To make our software perform well it should be error-free. If testing is done successfully it will remove all the errors from the software. In this article, we will discuss first the principles of testing and then we will discuss, the different types of testing.

Principles of Testing

- All the tests should meet the customer's requirements.
- To make our software testing should be performed by a third party.
- Exhaustive testing is not possible. As we need the optimal amount of testing based on the risk assessment of the application.
- All the tests to be conducted should be planned before implementing it
- It follows the Pareto rule(80/20 rule) which states that 80% of errors come from 20% of program components.
- Start testing with small parts and extend it to large parts.

- Types of Testing



Types of Software Testing

Different Types of Software Testing

1. **Manual Testing**
2. **Automation Testing**

1. Manual Testing

Manual testing is a technique to test the software that is carried out using the functions and features of an application. In manual software testing, a tester carries out tests on the software by following a set of predefined test cases. In this testing, testers make test cases for the codes, test the software, and give the final report about that software. Manual testing is time-consuming because it is done by humans, and there is a chance of human errors.

Advantages of Manual Testing:

- **Fast and accurate visual feedback:** It detects almost every bug in the software application and is used to test the dynamically changing GUI designs like layout, text, etc.
- **Less expensive:** It is less expensive as it does not require any high-level skill or a specific type of tool.
- **No coding is required:** No programming knowledge is required while using the black box testing method. It is easy to learn for the new testers.
- **Efficient for unplanned changes:** Manual testing is suitable in case of unplanned changes to the application, as it can be adopted easily.

2. Automation Testing

Automated Testing is a technique where the Tester writes scripts on their own and uses suitable Software or Automation Tool to test the software. It is an Automation Process of a Manual Process. It allows for executing repetitive tasks without the intervention of a Manual Tester.

Advantages of Automation Testing:

- **Simplifies Test Case Execution:** Automation testing can be left virtually unattended and thus it allows monitoring of the results at the end of the process. Thus, simplifying the overall test execution and increasing the efficiency of the application.
- **Improves Reliability of Tests:** Automation testing ensures that there is equal focus on all the areas of the testing, thus ensuring the best quality end product.
- **Increases amount of test coverage:** Using automation testing, more test cases can be created and executed for the application under test. Thus, resulting in higher test coverage and the detection of more bugs. This allows for the testing of more complex applications and more features can be tested.
- **Minimizing Human Interaction:** In automation testing, everything is automated from test case creation to execution thus there are no changes for human error due to neglect. This reduces the necessity for fixing glitches in the post-release phase.

Types of Manual Testing

1. **White Box Testing**
2. **Black Box Testing**
3. **Gray Box Testing**

1. 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 clear box testing or structural testing. White Box Testing is also known as transparent testing or 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.

Advantages of Whitebox Testing:

- **Thorough Testing**: White box testing is thorough as the entire code and structures are tested.
- **Code Optimization:** It results in the optimization of code removing errors and helps in removing extra lines of code.
- **Early Detection of Defects:** It can start at an earlier stage as it doesn't require any interface as in the case of black box testing.
- **Integration with SDLC:** White box testing can be easily started in the Software Development Life Cycle.
- **Detection of Complex Defects:** Testers can identify defects that cannot be detected through other testing techniques.

2. Black Box Testing

Black-box testing is a type of software testing in which the tester is not concerned with the internal knowledge or implementation details of the software but rather focuses on validating the functionality based on the provided specifications or requirements.

Advantages of Black Box Testing:

- The tester does not need to have more functional knowledge or programming skills to implement the Black Box Testing.
- It is efficient for implementing the tests in the larger system.
- Tests are executed from the user's or client's point of view.
- Test cases are easily reproducible.
- It is used to find the ambiguity and contradictions in the functional specifications.

3. Gray Box Testing

Gray Box Testing is a software testing technique that is a combination of the Black Box Testing technique and the White Box Testing technique.

1. In the Black Box Testing technique, the tester is unaware of the internal structure of the item being tested and in White Box Testing the internal structure is known to the tester.
2. The internal structure is partially known in Gray Box Testing.
3. This includes access to internal data structures and algorithms to design the test cases.

Advantages of Gray Box Testing:

1. **Clarity of goals:** Users and developers have clear goals while doing testing.
2. **Done from a user perspective:** Gray box testing is mostly done from the user perspective.
3. **High programming skills not required:** Testers are not required to have high programming skills for this testing.
4. **Non-intrusive:** Gray box testing is non-intrusive.
5. **Improved product quality:** Overall quality of the product is improved.

Types of Black Box Testing

1. **Functional Testing**
2. **Non-Functional Testing**

1. Functional Testing

Functional Testing is a type of Software Testing in which the system is tested against the functional requirements and specifications. Functional testing ensures that the requirements or specifications are properly satisfied by the application. This type of testing is particularly concerned with the result of processing. It focuses on the simulation of actual system usage but does not develop any system structure assumptions. The article focuses on discussing function testing.

Benefits of Functional Testing:

- **Bug-free product:** Functional testing ensures the delivery of a bug-free and high-quality product.
- **Customer satisfaction:** It ensures that all requirements are met and ensures that the customer is satisfied.
- **Testing focussed on specifications:** Functional testing is focussed on specifications as per customer usage.
- **Proper working of application:** This ensures that the application works as expected and ensures proper working of all the functionality of the application.
- **Improves quality of the product:** Functional testing ensures the security and safety of the product and improves the quality of the product.

2. Non-Functional Testing

Non-functional Testing is a type of Software Testing that is performed to verify the non-functional requirements of the application. It verifies whether the behavior of the system is as per the requirement or not. It tests all the aspects that are not tested in functional testing. Non-functional testing is a software testing technique that checks the non-functional attributes of the system. Non-functional testing is defined as a type of software testing to check non-functional aspects of a software application. It is designed to test the readiness of a system as per nonfunctional parameters which are never addressed by functional testing. Non-functional testing is as important as functional testing.

Benefits of Non-functional Testing

- **Improved performance:** Non-functional testing checks the performance of the system and determines the performance bottlenecks that can affect the performance.
- **Less time-consuming:** Non-functional testing is overall less time-consuming than the other testing process.
- **Improves user experience:** Non-functional testing like Usability testing checks how easily usable and user-friendly the software is for the users. Thus, focus on improving the overall user experience for the application.
- **More secure product:** As non-functional testing specifically includes security testing that checks the security bottlenecks of the application and how secure is the application against attacks from internal and external sources.

Types of Functional Testing

1. **Unit Testing**
2. **Integration Testing**
3. **System Testing**

1. Unit Testing

Unit testing is a method of testing individual units or components of a software application. It is typically done by developers and is used to ensure that the individual units of the software are working as intended. Unit tests are usually automated and are designed to test specific parts of the code, such as a particular function or method. Unit testing is done at the lowest level of the software development process, where individual units of code are tested in isolation.

Advantages of Unit Testing:

Some of the advantages of Unit Testing are listed below.

- It helps to identify bugs early in the development process before they become more difficult and expensive to fix.
- It helps to ensure that changes to the code do not introduce new bugs.
- It makes the code more modular and easier to understand and maintain.
- It helps to improve the overall quality and reliability of the software.

Note: *Some popular frameworks and tools that are used for unit testing include* ***JUnit****,* ***NUnit,*** *and* ***xUnit.***

- *It's important to keep in mind that Unit Testing is only one aspect of software testing and it should be used in combination with other types of testing such as integration testing, functional testing, and acceptance testing to ensure that the software meets the needs of its users.*
- *It focuses on the smallest unit of software design. In this, we test an individual unit or group of interrelated units. It is often done by the programmer by using sample input and observing its corresponding outputs.*

Example:

1. *In a program we are checking if the loop, method, or function is working fine.*
2. *Misunderstood or incorrect, arithmetic precedence.*
3. *Incorrect initialization.*

2. Integration Testing

Integration testing is a method of testing how different units or components of a software application interact with each other. It is used to identify and resolve any issues that may arise when different units of the software are combined. Integration testing is typically done after unit testing and before functional testing and is used to verify that the different units of the software work together as intended.

Different Ways of Performing Integration Testing:

Different ways of Integration Testing are discussed below.

- Top-down integration testing: It starts with the highest-level modules and differentiates them from lower-level modules.
- Bottom-up integration testing: It starts with the lowest-level modules and integrates them with higher-level modules.
- Big-Bang integration testing: It combines all the modules and integrates them all at once.
- Incremental integration testing: It integrates the modules in small groups, testing each group as it is added.

Advantages of Integrating Testing

- It helps to identify and resolve issues that may arise when different units of the software are combined.
- It helps to ensure that the different units of the software work together as intended.
- It helps to improve the overall reliability and stability of the software.
- It's important to keep in mind that Integration testing is essential for complex systems where different components are integrated.
- As with unit testing, integration testing is only one aspect of software testing and it should be used in combination with other types of testing such as unit testing, functional testing, and acceptance testing to ensure that the software meets the needs of its users.

The **objective** is to take unit-tested components and build a program structure that has been dictated by design. Integration testing is testing in which a group of components is combined to produce output.
Integration testing is of four types: (i) Top-down (ii) Bottom-up (iii) Sandwich (iv) Big-Bang
Example:

1. ***Black Box testing:*** *It is used for validation. In this, we ignore internal working mechanisms and focus on "what is the output?"*
2. ***White box testing:*** *It is used for verification. In this, we focus on internal mechanisms i.e. how the output is achieved.*

3. 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. It tests if the system meets the specified requirements and if it is suitable for delivery to the end-users. This type of testing is performed after the integration testing and before the acceptance testing.

System Testing *is a type of software testing that is performed on a completely integrated system to evaluate the compliance of the system with the corresponding requirements. In system testing, integration testing passed components are taken as input. The goal of integration testing is to detect any irregularity between the units that are integrated.*

Advantages of System Testing:

- The testers do not require more knowledge of programming to carry out this testing.
- It will test the entire product or software so that we will easily detect the errors or defects that cannot be identified during the unit testing and integration testing.
- The testing environment is similar to that of the real-time production or business environment.
- It checks the entire functionality of the system with different test scripts and also it covers the technical and business requirements of clients.
- After this testing, the product will almost cover all the possible bugs or errors and hence the development team will confidently go ahead with acceptance testing.

Types of Integration Testing

1. **Incremental Testing**
2. **Non-Incremental Testing**

1. Incremental Testing

Like development, testing is also a phase of SDLC (Software Development Life Cycle). Different tests are performed at different stages of the development cycle. Incremental testing is one of the testing approaches that is commonly used in the software field during the testing phase of integration testing which is performed after unit testing. Several stubs and drivers are used to test the modules one after one which helps in discovering errors and defects in the specific modules.

Advantages of Incremental Testing:

- Each module has its specific significance. Each one gets a role to play during the testing as they are incremented individually.
- Defects are detected in smaller modules rather than denoting errors and then editing and re-correcting large files.
- It's more flexible and cost-efficient as per requirements and scopes.
- The customer gets the chance to respond to each building.

There are 2 Types of Incremental Testing

1. **Top-down Integration Testing**
2. **Bottom-up Integration Testing**

1. Top-down Integration Testing

Top-down testing is a type of incremental integration testing approach in which testing is done by integrating or joining two or more modules by moving down from top to bottom through the control flow of the architecture structure. In these, high-level modules are tested first, and then low-level modules are tested. Then, finally, integration is done to ensure that the system is working properly. Stubs and drivers are used to carry out this project. This technique is used to increase or stimulate the behavior of Modules that are not integrated into a lower level.

Advantages Top Down Integration Testing

1. There is no need to write drivers.
2. Interface errors are identified at an early stage and fault localization is also easier.
3. Low-level utilities that are not important are not tested well and high-level testers are tested well in an appropriate manner.
4. Representation of test cases is easier and simpler once Input-Output functions are added.

2. Bottom-up Integration Testing

Bottom-up Testing is a type of incremental integration testing approach in which testing is done by integrating or joining two or more modules by moving upward from bottom to top through the control flow of the architecture structure. In these, low-level modules are tested first, and then high-level modules are tested. This type of testing or approach is also known as inductive reasoning and is used as a synthesis synonym in many cases. Bottom-up testing is user-friendly testing and results in an increase in overall software development. This testing results in high success rates with long-lasting results.

Advantages of Bottom-up Integration Testing:

- It is easy and simple to create and develop test conditions.
- It is also easy to observe test results.
- It is not necessary to know about the details of the structural design.
- Low-level utilities are also tested well and are also compatible with the object-oriented structure.

Types of Non-functional Testing

1. **Performance Testing**
2. **Usability Testing**
3. **Compatibility Testing**

1. Performance Testing

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

Performance testing is a type of software testing that focuses on evaluating the performance and scalability of a system or application. The goal of performance testing is to identify bottlenecks, measure system performance under various loads and conditions, and ensure that the system can handle the expected number of users or transactions.

Advantages of Performance Testing:

- Performance testing ensures the speed, load capability, accuracy, and other performances of the system.
- It identifies, monitors, and resolves the issues if anything occurs.
- It ensures the great optimization of the software and also allows many users to use it at the same time.
- It ensures the client as well as the end-customer's satisfaction. Performance testing has several advantages that make it an important aspect of software testing:
- **Identifying bottlenecks**: Performance testing helps identify bottlenecks in the system such as slow database queries, insufficient memory, or network congestion. This helps developers optimize the system and ensure that it can handle the expected number of users or transactions.

2. Usability Testing

You design a product (say a refrigerator) and when it becomes completely ready, you need a potential customer to test it to check it working. To understand whether the machine is ready to come on the market, potential customers test the machines. Likewise, the best example of usability testing is when the software also undergoes various testing processes which is performed by potential users before launching into the market. It is a part of the software development lifecycle (SDLC).

Advantages and Disadvantages of Usability Testing:

Usability testing is preferred to evaluate a product or service by testing it with the proper users. In Usability testing, the development and design teams will use to identify issues before coding and the result will be earlier issues will be solved. During a Usability test, you can,

- Learn if participants will be able to complete the specific task completely.
- identify how long it will take to complete the specific task.
- Gives excellent features and functionalities to the product
- Improves user satisfaction and fulfills requirements based on user feedback
- The product becomes more efficient and effective

3. Compatibility Testing

Compatibility testing is software testing that comes under the non functional testing category, and it is performed on an application to check its compatibility (running capability) on different platforms/environments. This testing is done only when the application becomes stable. This means simply this compatibility test aims to check the developed software application functionality on various software, hardware platforms, networks browser etc. This compatibility testing is very important in product production and implementation point of view as it is performed to avoid future issues regarding compatibility.

Advantages of Compatibility Testing:

- It ensures complete customer satisfaction.
- It provides service across multiple platforms.
- Identifying bugs during the development process.

There are 4 Types of Performance Testing

1. **Load Testing**
2. **Stress Testing**
3. **Scalability Testing**
4. **Stability Testing**

1. Load Testing

Load testing determines the behavior of the application when multiple users use it at the same time. It is the response of the system measured under varying load conditions.

1. The load testing is carried out for normal and extreme load conditions.
2. Load testing is a type of performance testing that simulates a real-world load on a system or application to see how it performs under stress.
3. The goal of load testing is to identify bottlenecks and determine the maximum number of users or transactions the system can handle.
4. It is an important aspect of software testing as it helps ensure that the system can handle the expected usage levels and identify any potential issues before the system is deployed to production.

Advantages of Load Testing:

Load testing has several advantages that make it an important aspect of software testing:

1. **Identifying bottlenecks:** Load testing helps identify bottlenecks in the system such as slow database queries, insufficient memory, or network congestion. This helps developers optimize the system and ensure that it can handle the expected number of users or transactions.
2. **Improved scalability:** By identifying the system's maximum capacity, load testing helps ensure that the system can handle an increasing number of users or transactions over time. This is particularly important for web-based systems and applications that are expected to handle a high volume of traffic.
3. **Improved reliability:** Load testing helps identify any potential issues that may occur under heavy load conditions, such as increased error rates or slow response times. This helps ensure that the system is reliable and stable when it is deployed to production.

2. Stress Testing

In Stress Testing, we give unfavorable conditions to the system and check how it perform in those conditions.

Example:

1. *Test cases that require maximum memory or other resources are executed.*
2. *Test cases that may cause thrashing in a virtual operating system.*
3. *Test cases that may cause excessive disk requirement Performance Testing.*

It is designed to test the run-time performance of software within the context of an integrated system. It is used to test the speed and effectiveness of the program. It is also called load testing. In it, we check, what is the performance of the system in the given load.

Example:

Checking several processor cycles.

3. Scalability Testing

Scalability Testing is a type of non-functional testing in which the performance of a software application, system, network, or process is tested in terms of its capability to scale up or scale down the number of user request load or other such performance attributes. It can be carried out at a hardware, software or database level. Scalability Testing is defined as the ability of a network, system, application, product or a process to perform the function correctly when changes are made in the size or volume of the system to meet a growing need. It ensures that a software product can manage the scheduled increase in user traffic, data volume, transaction counts frequency, and many other things. It tests the system, processes, or database's ability to meet a growing need.

Advantages of Scalability Testing:

- It provides more accessibility to the product.
- It detects issues with web page loading and other performance issues.
- It finds and fixes the issues earlier in the product which saves a lot of time.
- It ensures the end-user experience under the specific load. It provides customer satisfaction.
- It helps in effective tool utilization tracking.

4. Stability Testing

Stability Testing *is a type of Software Testing to checks the quality and behavior of the software under different environmental parameters. It is defined as the ability of the product to continue to function over time without failure.*

It is a Non-functional Testing technique that focuses on stressing the software component to the maximum. Stability testing is done to check the efficiency of a developed product beyond normal operational capacity which is known as break point. It has higher significance in error handling, software reliability, robustness, and scalability of a product under heavy load rather than checking the system behavior under normal circumstances.

Stability testing assesses stability problems. This testing is mainly intended to check whether the application will crash at any point in time or not.

Advantages of Stability Testing:

1. It gives the limit of the data that a system can handle practically.
2. It provides confidence on the performance of the system.
3. It determines the stability and robustness of the system under load.
4. Stability testing leads to a better end-user experience.

Other Types of Testing

1. **Smoke Testing**
2. **Sanity Testing**
3. **Regression Testing**
4. **Acceptance Testing**
5. **User Acceptance Testing**
6. **Exploratory Testing**
7. **Adhoc Testing**
8. **Security Testing**
9. **Globalization Testing**
10. **Regression Testing**
11. **Smoke Testing**
12. **Alpha Testing**
13. **Beta Testing**
14. **Object-Oriented Testing**

1. Smoke Testing

Smoke Testing is done to make sure that the software under testing is ready or stable for further testing It is called a smoke test as the testing of an initial pass is done to check if it did not catch fire or smoke in the initial switch-on.

Example:

If the project has 2 modules so before going to the module make sure that module 1 works properly.

Advantages of Smoke Testing:

1. Smoke testing is easy to perform.
2. It helps in identifying defects in the early stages.
3. It improves the quality of the system.
4. Smoke testing reduces the risk of failure.
5. Smoke testing makes progress easier to access.

2. Sanity Testing

It is a **subset** of regression testing. Sanity testing is performed to ensure that the code changes that are made are working properly. Sanity testing is a stoppage to check whether testing for the build can proceed or not. The focus of the team during the sanity testing process is to validate the functionality of the application and not detailed testing. Sanity testing is generally performed on a build where the production deployment is required immediately like a critical bug fix.

Advantages of Sanity Testing:

- Sanity testing helps to quickly identify defects in the core functionality.
- It can be carried out in less time as no documentation is required for sanity testing.
- If the defects are found during sanity testing, the project is rejected which is helpful in saving time for execution of regression tests.
- This testing technique is not so expensive when compared to another type of testing.
- It helps to identify the dependent missing objects.

3. Regression Testing

The process of testing the modified parts of the code and the parts that might get affected due to the modifications ensures that no new errors have been introduced in the software after the modifications have been made. Regression means the return of something and in the software field, it refers to the return of a bug.

Advantages of Regression Testing:

- It ensures that no new bugs have been introduced after adding new functionalities to the system.
- 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.
- It helps to maintain the quality of the source code.

4. Acceptance Testing

Acceptance testing is done by the customers to check whether the delivered products perform the desired tasks or not, as stated in the requirements. We use Object-Oriented Testing for discussing test plans and for executing the projects.

Advantages of Acceptance Testing:

1. This testing helps the project team to know the further requirements of the users directly as it involves the users for testing.
2. Automated test execution.
3. It brings confidence and satisfaction to the clients as they are directly involved in the testing process.
4. It is easier for the user to describe their requirement.
5. It covers only the Black-Box testing process and hence the entire functionality of the product will be tested.

5. User Acceptance Testing

User Acceptance Testing is a testing methodology where clients/end users participate in product testing to validate the product against their requirements. It is done at the client's site on the developer's site. For industries such as medicine or aerospace, contractual and regulatory compliance testing, and operational

acceptance tests are also performed as part of user acceptance tests. UAT is context-dependent and UAT plans are prepared based on requirements and are not required to perform all kinds of user acceptance tests and are even coordinated and contributed by the testing team.

6. Exploratory Testing

Exploratory Testing is a type of software testing in which the tester is free to select any possible methodology to test the software. It is an unscripted approach to software testing. In exploratory testing, software developers use their learning, knowledge, skills, and abilities to test the software developed by themselves. Exploratory testing checks the functionality and operations of the software as well as identifies the functional and technical faults in it. Exploratory testing aims to optimize and improve the software in every possible way.

Advantages of Exploratory Testing:

- **Less preparation required:** It takes no preparation as it is an unscripted testing technique.
- **Finds critical defects:** Exploratory testing involves an investigation process that helps to find critical defects very quickly.
- **Improves productivity:** In exploratory testing, testers use their knowledge, skills, and experience to test the software. It helps to expand the imagination of the testers by executing more test cases, thus enhancing the overall quality of the software.

7. Adhoc Testing

Adhoc testing is a type of software testing that is performed informally and randomly after the formal testing is completed to find any loophole in the system. For this reason, it is also known as Random or Monkey testing. Adhoc testing is not performed in a structured way so it is not based on any methodological approach. That's why Adhoc testing is a type of Unstructured Software Testing.

Advantages of Adhoc testing:

- The errors that can not be identified with written test cases can be identified by Adhoc testing.
- It can be performed within a very limited time.
- Helps to create unique test cases.
- This test helps to build a strong product that is less prone to future problems.
- This testing can be performed at any time during Softthe ware Development Life Cycle Process (SDLC)

8. Security Testing

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

Advantages of Security Testing:

1. Identifying vulnerabilities: Security testing helps identify vulnerabilities in the system that could be exploited by attackers, such as weak passwords, unpatched software, and misconfigured systems.
2. Improving system security: Security testing helps improve the overall security of the system by identifying and fixing vulnerabilities and potential threats.
3. Ensuring compliance: Security testing helps ensure that the system meets relevant security standards and regulations, such as HIPAA, PCI DSS, and SOC2.

9. Globalization Testing

Globalization Testing is a type of software testing that is performed to ensure the system or software application can function independently of the geographical and cultural environment. It ensures that the application can be used all over the world and accepts all language texts. Nowadays with the increase in various technologies, every software product is designed in such a way that it is a globalized software product.

Benefits of Globalization Testing:

- **Helps to create scalable products:** It makes the software product more flexible and scalable.
- **Save time:** It saves overall time and effort for software testing.

- **Reduce time for localization testing:** Globalization testing helps to reduce the time and cost of localization testing.

10. Regression Testing

Regression testing is a method of testing that is used to ensure that changes made to the software do not introduce new bugs or cause existing functionality to break. It is typically done after changes have been made to the code, such as bug fixes or new features, and is used to verify that the software still works as intended.

Regression testing can be performed in different ways, such as:

- **Retesting**: This involves testing the entire application or specific functionality that was affected by the changes.
- **Re–execution**: This involves running a previously executed test suite to ensure that the changes did not break any existing functionality.
- **Comparison**: This involves comparing the current version of the software with a previous version to ensure that the changes did not break any existing functionality.

Advantages of Regression Testing

- It helps to ensure that changes made to the software do not introduce new bugs or cause existing functionality to break.
- It helps to ensure that the software continues to work as intended after changes have been made.
- It helps to improve the overall reliability and stability of the software.
- It's important to keep in mind that regression testing is an ongoing process that should be done throughout the software development
- lifecycle to ensure that the software continues to work as intended. It should be automated as much as possible to save time and resources. Additionally, it's important to have a well-defined regression test suite that covers

Every time a new module is added leads to changes in the program. This type of testing makes sure that the whole component works properly even after adding components to the complete program.

Example:

In school records, suppose we have module staff, students, and finance combining these modules and checking if the integration of these modules works fine in regression testing.

11. Smoke Testing

Smoke Testing is done to make sure that the software under testing is ready or stable for further testing It is called a smoke test as the testing of an initial pass is done to check if it did not catch fire or smoke in the initial switch-on.

Example:

If the project has 2 modules so before going to the module make sure that module 1 works properly.

12. Alpha Testing

Alpha testing is a type of validation testing. It is a type of acceptance testing that is done before the product is released to customers. It is typically done by QA people.

Example:

When software testing is performed internally within the organisation.

13. Beta Testing

The beta test is conducted at one or more customer sites by the end-user of the software. This version is released for a limited number of users for testing in a real-time environment.

Example:

When software testing is performed for the limited number of people.

14. Object-Oriented Testing

Object-Oriented Testing testing is a combination of various testing techniques that help to verify and validate object-oriented software. This testing is done in the following manner:

- Testing of Requirements,
- Design and Analysis of Testing,
- Testing of Code,
- Integration testing,
- System testing,

- User Testing.

Advantages of Software Testing

1. Improved software quality and reliability.
2. Early identification and fixing of defects.
3. Improved customer satisfaction.
4. Increased stakeholder confidence.
5. Reduced maintenance costs.
6. Customer Satisfaction
7. Cost Effective
8. Quality Product
9. Low Failure
10. Bug-Free Application
11. Security
12. Speed Up the Development Process
13. Early Defect Detection
14. Reliable Product

Disadvantages of Software Testing

- Time-consuming and adds to the project cost.
- This can slow down the development process.
- Not all defects can be found.
- Can be difficult to fully test complex systems.
- Potential for human error during the testing process.

General Categories of Bug Taxonomy

Bug taxonomy typically encompasses a range of general categories, each focusing on specific aspects of the software being tested.

These categories can vary depending on the project and industry, but some common examples include:

Functionality

This category involves bugs related to the expected behavior and features of the software. It covers issues such as incorrect calculations, missing functionalities, and inconsistencies in user interface elements.

Performance

Bugs in this category affect the software's speed, responsiveness, resource usage, and overall performance. Examples include slow program execution, poor responsiveness, and inefficient memory management.

Usability

Usability-related bugs impact the user experience, including aspects like intuitiveness, accessibility, and ease of learning. Such bugs may involve unclear instructions, non-intuitive navigation, or inconsistent behavior across different devices.

Security

Security bugs encompass vulnerabilities, privacy breaches, and potential exploits within the software. These bugs can lead to unauthorized access, data breaches, or compromised user privacy.

Compatibility

Compatibility issues arise when the software fails to function properly across different platforms, browsers, or hardware configurations. These bugs may result in inconsistent behavior, layout distortions, or functional limitations.

Consequences of Bugs

The consequences of a bug can be measured in terms of human, rather than machine, Some consequences of a bug on a scale of one to ten are:
1. Mild: The symptoms of the bug offend us gently; a misspelled output or a misaligned printout
2. Moderate: Outputs are misleading or redundant. The bug impacts the systems performance
3. Annoying: The Systems behaviour, because of the bug is dehumanizing
Example: Names are truncated or arbitrarily modified
4. Disturbing: It refuses to handle legitimate (authorized/legal) transactions.
Example: The ATM wont give money. Credit card is declared invalid
5. Serious: It loses track of its transaction. Not just the transaction itself but the fact that the transaction occurred and accountability is lost
6. Very Serious: The bug causes the system to do the wrong transactions. Instead of gaining a pay check, the system credits it to another account or converts deposits to withdrawals
7. Extreme: The problems are not limited to a few users or to few transaction types. They are frequent and arbitrary, (instead of occasionally infrequent) or for unusual case
8. Intolerable: Long term unrecoverable corruption of the database occurs and the corruption is not easily discovered. Serious consideration is given to shutting the system down
9. Catastrophic: The decision to shut down is taken out of our hands because the system fails
10. Infectious: What can be worse than a failed system? One that corrupts other systems even though it does not fail in itself; that erodes the social physical environment; that melts reactors and starts a war

Unit – IV

Structural Testing (White Box Testing)

Structural Testing

In this section, we are going to understand **structural testing**, which is a significant part of **Software testing**.

And we also learn about its needs, **types of structural testing, tools compatible for Structural testing, advantage, and disadvantage.**

Introduction of Structural Testing

Another type of **software testing** technique is **Structural testing**, which is used to test the internal design of the software or structure of the coding for the particular software.

In this testing, the development team members are included in the testing team to execute the software's internal design. The working of structural testing is opposite to **Behavioral testing**.

In other words, we can say that structural testing tests the different features of an application based on its types.

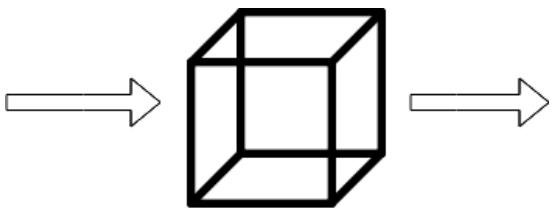
Structural testing is also known as white-box testing, **glass box testing**, and **clear-box testing**. Developers mostly implement it to identify the issue and fix them quickly.

The structural testing process requires an in-depth knowledge of the programming language and is opposite to Functional Testing.

The knowledge of the code's internal executions and how the software is implemented is a necessity for the test engineer to implement the structural testing.

Throughout the structural testing, the test engineer intends on how the software performs, and it can be used at all levels of testing.

Test cases for white box testing are derived from the design phase of the software development lifecycle. Data flow testing, control flow testing, path testing, branch testing, statement and decision coverage all these techniques used by white box testing as a guideline to create an error-free software.



Whitebox Testing

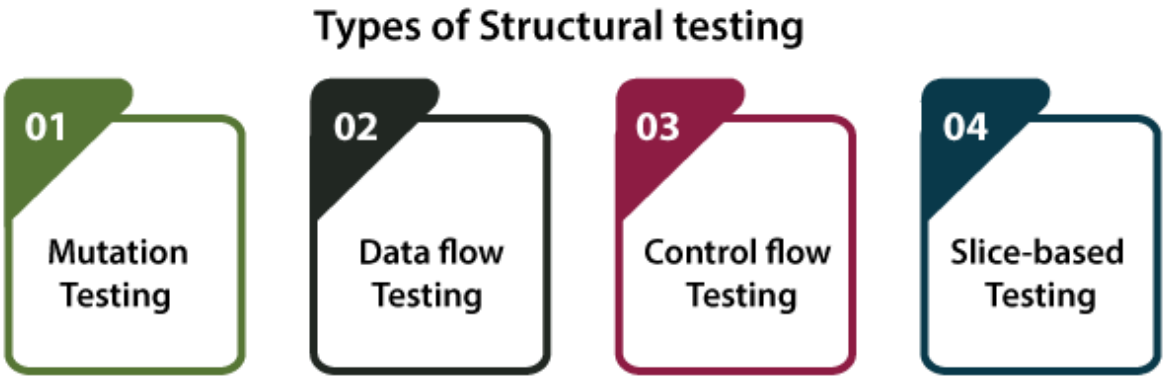
White box testing follows some working steps to make testing manageable and easy to understand what the next task to do. There are some basic steps to perform white box testing.

For example, the coverage of menu options or significant business transactions could be the system's structural element or acceptance testing.

Types of Structural Testing

Structural testing is divided into four different categories, which are as follows:

- **Mutation testing**
- **Data flow testing**
- **Control flow testing**
- **Slice-based testing**



Mutation testing

- It is used to check the quality of the test case that should fail the mutant code.
- Mutation testing involves the development of new tests to be implemented on the software for its testing process.
- When we identify various errors, it implies that either the program is correct or the test case is inefficient in locating the fault.
- In the mutation testing, the developers make small modifications to the previously accessible software tests and generate a mutant of the old software test.
- It used to cause an error in the program, which implies that the mutation testing is performed to evaluate the test case's productivity.

Data flow testing

- It is a group of testing approaches used to observe the control flow of programs to discover the sequence of variables as per the series of events.

- It implements a control flow graph and analysis the points where the codes can change the data.
- If we execute the data flow testing technique, the information is kept safe and unchanged during the code's implementation.

Control flow testing

- The **control flow testing** is the basic model of **Structural testing**.
- It is to check the implementation order of commands or statements of the code over a control structure.
- In the control flow testing, a specific part of an extensive program is selected by the test engineer to set the testing path.
- Generally, the control flow testing technique is used in unit testing.
- In this testing, the entire test is based on how the control is executed during the code.
- The complete information of all the software's features and logic is necessary to execute the control flow testing.

Slice-based testing

- It was initially created and established to keep the software.
- The basic idea is to sort the complete code into small chunks and then evaluate each portion carefully.
- The slice-based testing is very beneficial for the maintenance of the software along with fixing the application too.

Structural Testing Tools

Like other testing has their tools, structural testing also contains some open-source and commercial tools that have their functionality.

Some of the most commonly used tools for structural testing are as follows:

- **Cucumber**
- **JBehave**
- **Cfix**
- **JUnit**

Advantages and Disadvantages of Structural Testing

Below are the benefits and drawback of structural testing:

Advantages of Structural Testing

The benefits of Structural testing are as follows:

- Structural testing does not require a lot of manual work as it is an automated process.
- Structural testing is not a time-consuming process.
- All the early defects can easily be identified
- It removes the dead code (extra code) or statements easily.
- It provides easy coding and implementation.
- It delivers detailed testing of the software.

Disadvantages of Structural Testing

The **drawback** of the structural testing are as follows:

- To perform structural testing, in-depth knowledge of programming languages is required.
- Even though structural testing is automatic, it might turn out very difficult because it involves training in the tool used for testing.
- It is expensive in respect of money because sometimes resources are necessary to efficiently perform structural testing.
- There is also a small chance that some commands, statements or branches could be missed unintentionally.

Overview

In this tutorial, we have understood structural testing, the type of structural testing, advantages and disadvantages.

After learning all the specific topics, we can easily conclude that Structural testing, which is also called **White-box testing, glass box testing, and clear box testing** is used to verify the structure of code.

And apart from that, we can say that while executing the different types of software testing, there is no guarantee of 100% efficiency of the product. Therefore, it is always helpful if we associate different categories of testing and methods.

The various structural testing types, which we have looked at, such as data flow testing, mutation testing, slice-based testing, and control flow testing, could be copied back to errors such as:

- Mutation testing (using the wrong operator).

- Data flow testing (referencing a variable before using it).

In case someone is looking to use the structural testing methods, they need to consider both the benefits and drawbacks of the structural testing.

Furthermore, they need to take care of the fact that structural testing is implemented successfully.

Functional Testing (Black Box Testing)

Functional Testing

Before proceeding to functional testing, we should know about the testing, what testing is?

What is testing?

In simple terms, the testing is to compare the actual result with the expected result. Testing is done to identify whether all the function is working as expectations.

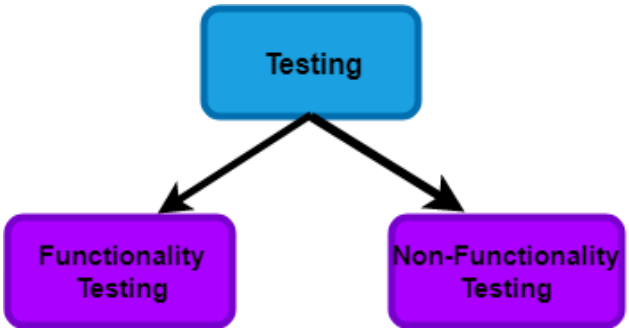
What is Software Testing?

Software testing is a technique to check whether the actual result matches the expected result and to ensure that the software has not any defect or bug.

Software testing ensures that the application has not any defect or the requirement is missing to the actual need. Either manual or automation testing can do software testing.

Software testing also defines as verification of application under test (AUT).

There are two types of testing:



Functional Testing:

It is a type of software testing which is used to verify the functionality of the software application, whether the function is working according to the requirement specification. In functional testing, each function tested by giving the value, determining the output, and verifying the actual output with the expected value. Functional testing performed as black-box testing which is presented to confirm that the functionality of an application or system behaves as we are expecting. It is done to verify the functionality of the application.

Functional testing also called as black-box testing, because it focuses on application specification rather than actual code. Tester has to test only the program rather than the system.

Goal of functional testing

The purpose of the functional testing is to check the primary entry function, necessarily usable function, the flow of screen GUI. Functional testing displays the error message so that the user can easily navigate throughout the application.

What is the process of functional testing?

Testers follow the following steps in the functional testing:

- Tester does verification of the requirement specification in the software application.
- After analysis, the requirement specification tester will make a plan.
- After planning the tests, the tester will design the test case.
- After designing the test, case tester will make a document of the traceability matrix.
- The tester will execute the test case design.
- Analysis of the coverage to examine the covered testing area of the application.
- Defect management should do to manage defect resolving.

What to test in functional testing? Explain

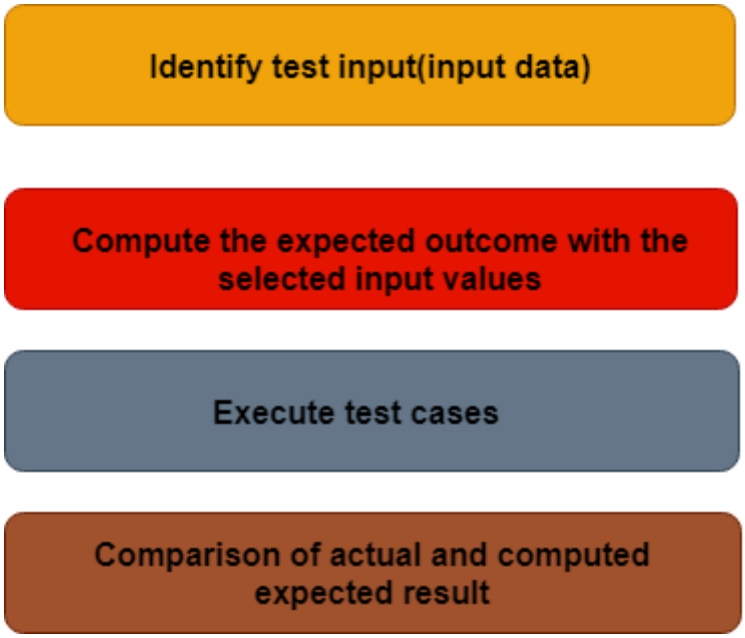
The main objective of functional testing is checking the functionality of the software system. It concentrates on:

- **Basic Usability:** Functional Testing involves the usability testing of the system. It checks whether a user can navigate freely without any difficulty through screens.
- **Accessibility:** Functional testing test the accessibility of the function.
- **Mainline function:** It focuses on testing the main feature.
- **Error Condition:** Functional testing is used to check the error condition. It checks whether the error message displayed.

Explain the complete process to perform functional testing.

There are the following steps to perform functional testing:

- There is a need to understand the software requirement.
- Identify test input data
- Compute the expected outcome with the selected input values.
- Execute test cases
- Comparison between the actual and the computed result

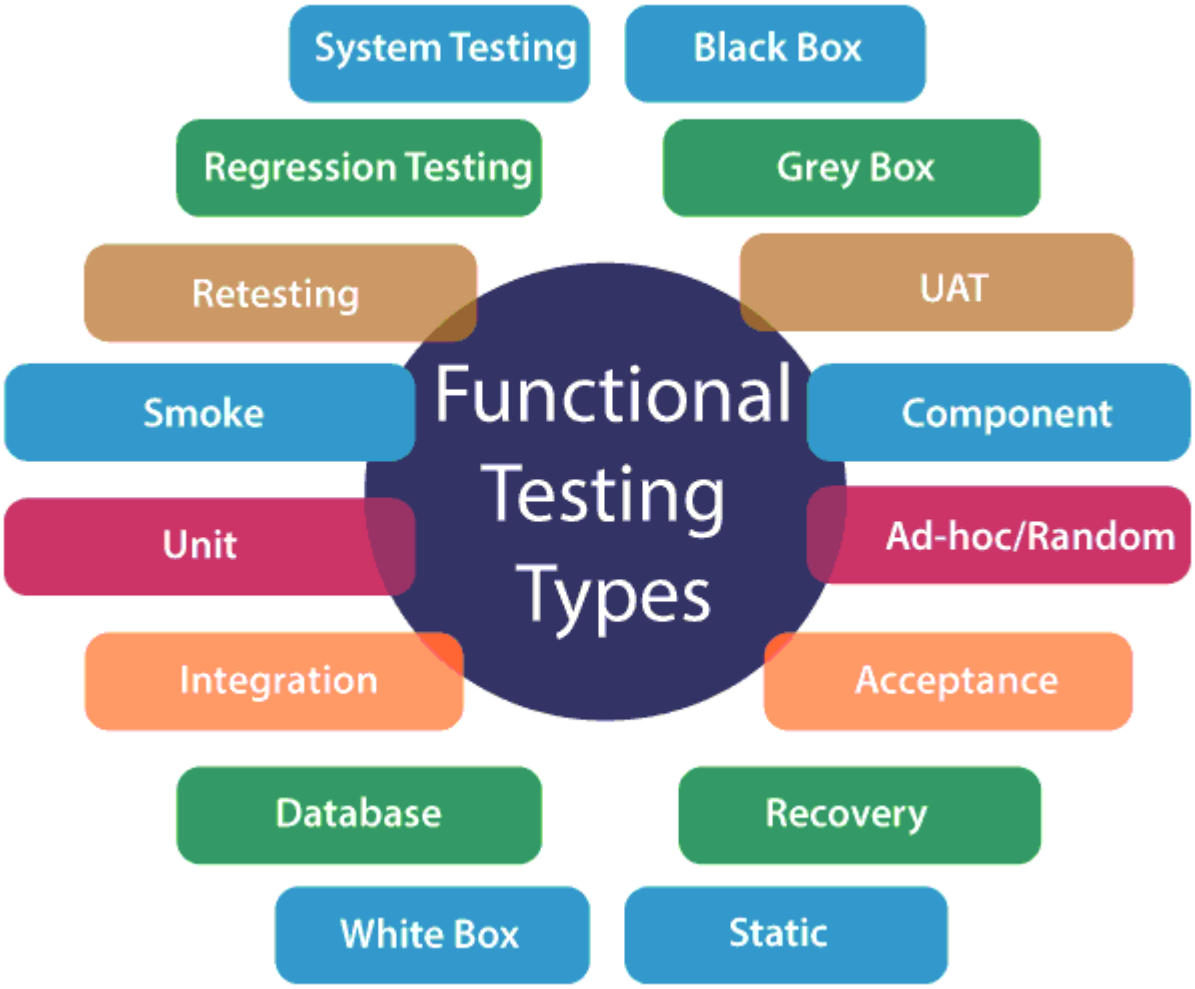


Explain the types of functional testing.

The main objective of functional testing is to test the functionality of the component.

Functional testing is divided into multiple parts.

Here are the following types of functional testing.



Unit Testing: Unit testing is a type of software testing, where the individual unit or component of the software tested. Unit testing, examine the different part of the application, by unit testing functional testing also done, because unit testing ensures each module is working correctly.

The developer does unit testing. Unit testing is done in the development phase of the application.

Smoke Testing: Functional testing by smoke testing. Smoke testing includes only the basic (feature) functionality of the system. Smoke testing is known as "***Build Verification Testing***." Smoke testing aims to ensure that the most important function work.

For example, Smoke testing verifies that the application launches successfully will check that GUI is responsive.

Sanity Testing: Sanity testing involves the entire high-level business scenario is working correctly. Sanity testing is done to check the functionality/bugs fixed. Sanity testing is little advance than smoke testing.

For example, login is working fine; all the buttons are working correctly; after clicking on the button navigation of the page is done or not.

Regression Testing: This type of testing concentrate to make sure that the code changes should not side effect the existing functionality of the system. Regression testing specifies when bug arises in the system after fixing the bug, regression testing concentrate on that all parts are working or not. Regression testing focuses on is there any impact on the system.

Integration Testing: Integration testing combined individual units and tested as a group. The purpose of this testing is to expose the faults in the interaction between the integrated units.

Developers and testers perform integration testing.

White box testing: White box testing is known as Clear Box testing, code-based testing, structural testing, extensive testing, and glass box testing, transparent box testing. It is a software testing method in which the internal structure/design/ implementation tested known to the tester.

The white box testing needs the analysis of the internal structure of the component or system.

Black box testing: It is also known as behavioral testing. In this testing, the internal structure/ design/ implementation not known to the tester. This type of testing is functional testing. Why we called this type of testing is black-box testing, in this testing tester, can't see the internal code.

For example, A tester without the knowledge of the internal structures of a website tests the web pages by using the web browser providing input and verifying the output against the expected outcome.

User acceptance testing: It is a type of testing performed by the client to certify the system according to requirement. The final phase of testing is user acceptance testing before releasing the software to the market or production environment. UAT is a kind of black-box testing where two or more end-users will involve.

Retesting: Retesting is a type of testing performed to check the test cases that were unsuccessful in the final execution are successfully pass after the defects fixed. Usually, tester assigns the bug when they find it while testing the product or its component. The bug allocated to a developer, and he fixes it. After fixing, the bug is assigned to a tester for its verification. This testing is known as retesting.

Database Testing: Database testing is a type of testing which checks the schema, tables, triggers, etc. of the database under test. Database testing may involve creating complex queries to load/stress test the database and check its responsiveness. It checks the data integrity and consistency.

Example: let us consider a banking application whereby a user makes a transaction. Now from database testing following, things are important. They are:

- Application store the transaction information in the application database and displays them correctly to the user.
- No information lost in this process
- The application does not keep partially performed or aborted operation information.
- The user information is not allowed individuals to access by the

Ad-hoc testing: Ad-hoc testing is an informal testing type whose aim is to break the system. This type of software testing is unplanned activity. It does not follow any test design to create the test cases. Ad-hoc testing is done randomly on any part of the application; it does not support any structured way of testing.

Recovery Testing: Recovery testing is used to define how well an application can recover from crashes, hardware failure, and other problems. The purpose of recovery testing is to verify the system's ability to recover from testing points of failure.

Static Testing: Static testing is a software testing technique by which we can check the defects in software without actually executing it. Static testing is done to avoid errors in the early stage of the development as it is easier to find failure in the early stages. Static testing used to detect the mistakes that may not found in dynamic testing.

Why we use static testing?

Static testing helps to find the error in the early stages. With the help of static testing, this will reduce the development timescales. It reduces the testing cost and time. Static testing also used for development productivity.

Component Testing: Component Testing is also a type of software testing in which testing is performed on each component separately without integrating with other parts. Component testing is also a type of black-box testing. Component testing also referred to as Unit testing, program testing, or module testing.

Grey Box Testing: Grey Box Testing defined as a combination of both white box and black-box testing. Grey Box testing is a testing technique which performed with limited information about the internal functionality of the system.

What are the functional testing tools?

The functional testing can also be executed by various apart from manual testing. These tools simplify the process of testing and help to get accurate and useful results.

It is one of the significant and top-priority based techniques which were decided and specified before the development process.

Tools Used for Black Box Testing:

1. **Appium**
2. **Selenium**
3. **Microsoft Coded UI**
4. **Applitools**
5. **HP QTP.**

What are the advantages of Functional Testing?

Advantages of functional testing are:

- It produces a defect-free product.
- It ensures that the customer is satisfied.
- It ensures that all requirements met.
- It ensures the proper working of all the functionality of an application/software/product.
- It ensures that the software/ product work as expected.
- It ensures security and safety.
- It improves the quality of the product.

Example: Here, we are giving an example of banking software. In a bank when money transferred from bank A to bank B. And the bank B does not receive the correct amount, the fee is applied, or the money not converted into the correct currency, or incorrect transfer or bank A does not receive statement advice from bank B that the payment has received. These issues are critical and can be avoided by proper functional testing.

What are the disadvantages of functional testing?

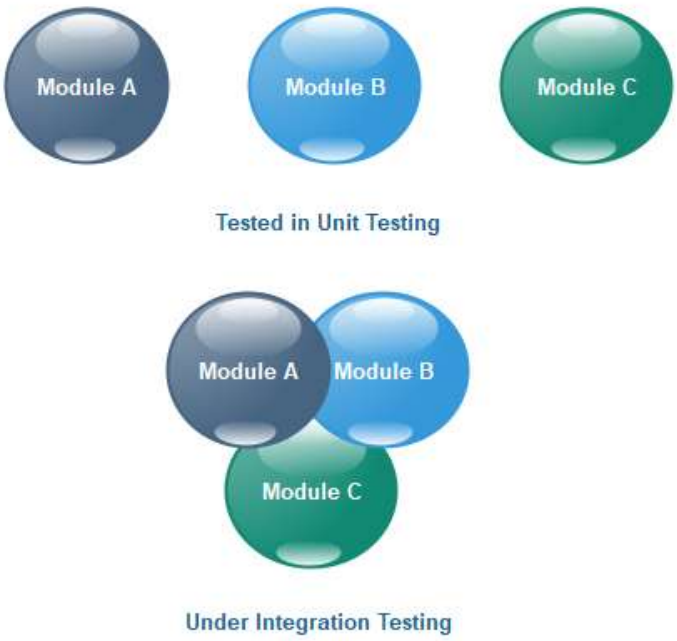
Disadvantages of functional testing are:

- Functional testing can miss a critical and logical error in the system.
- This testing is not a guarantee of the software to go live.
- The possibility of conducting redundant testing is high in functional testing.

Integration testing

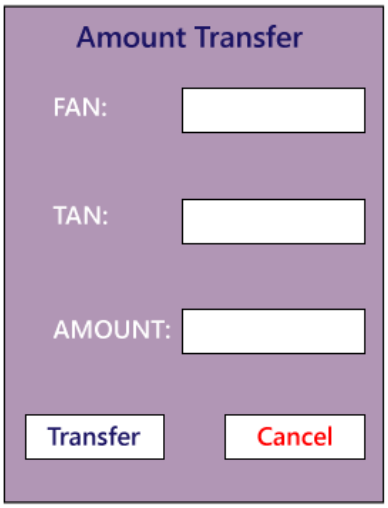
Integration testing is the second level of the software testing process comes after unit testing. In this testing, units or individual components of the software are tested in a group. The focus of the integration testing level is to expose defects at the time of interaction between integrated components or units.

Unit testing uses modules for testing purpose, and these modules are combined and tested in integration testing. The Software is developed with a number of software modules that are coded by different coders or programmers. The goal of integration testing is to check the correctness of communication among all the modules.



Once all the components or modules are working independently, then we need to check the data flow between the dependent modules is known as **integration testing**.

Let us see one sample example of a banking application, as we can see in the below image of amount transfer.



- First, we will login as a user **P** to amount transfer and send Rs200 amount, the confirmation message should be displayed on the screen as **amount transfer successfully**. Now logout as P and login as user **Q** and go to amount balance page and check for a balance in that account = Present balance + Received Balance. Therefore, the integration test is successful.
- Also, we check if the amount of balance has reduced by Rs200 in P user account.
- Click on the transaction, in P and Q, the message should be displayed regarding the data and time of the amount transfer.

Guidelines for Integration Testing

- We go for the integration testing only after the functional testing is completed on each module of the application.
- We always do integration testing by picking module by module so that a proper sequence is followed, and also we don't miss out on any integration scenarios.
- First, determine the test case strategy through which executable test cases can be prepared according to test data.
- Examine the structure and architecture of the application and identify the crucial modules to test them first and also identify all possible scenarios.
- Design test cases to verify each interface in detail.
- Choose input data for test case execution. Input data plays a significant role in testing.
- If we find any bugs then communicate the bug reports to developers and fix defects and retest.
- Perform **positive and negative integration testing**.

Here **positive** testing implies that if the total balance is Rs15, 000 and we are transferring Rs1500 and checking if the amount transfer works fine. If it does, then the test would be a pass.

And **negative testing** means, if the total balance is Rs15, 000 and we are transferring Rs20, 000 and check if amount transfer occurs or not, if it does not occur, the test is a pass. If it happens, then there is a bug in the code, and we will send it to the development team for fixing that bug.

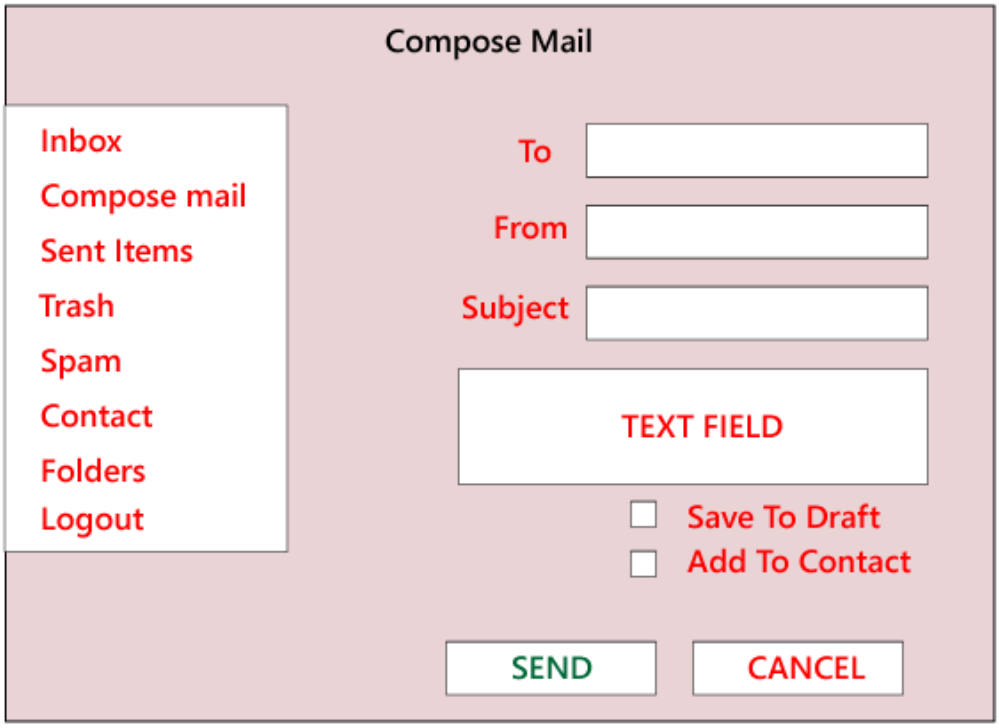
For example: In the Gmail application, the **Source** could be **Compose**, **Data** could be **Email** and the **Destination** could be the **Inbox**.

Example of integration testing

Let us assume that we have a **Gmail** application where we perform the integration testing.

First, we will do **functional testing** on **the login page**, which includes the various components such as **username, password, submit, and cancel** button. Then only we can perform integration testing.

The different integration scenarios are as follows:

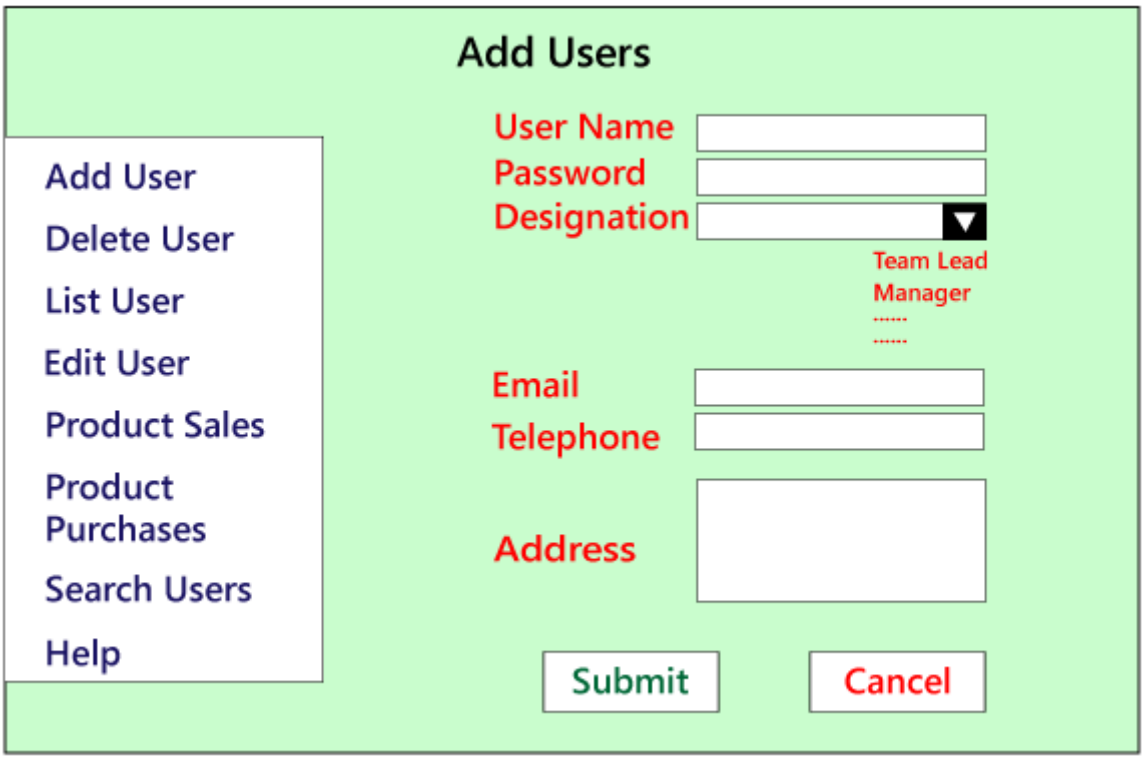


Scenarios1:

- First, we login as **P** users and click on the **Compose** mail and performing the functional testing for the specific components.
- Now we click on the **Send** and also check for **Save Drafts**.
- After that, we send a **mail** to **Q** and verify in the **Send Items** folder of P to check if the send mail is there.
- Now, we will **log out** as P and login as Q and move to the **Inbox** and verify that if the mail has reached.

Secanrios2: We also perform the integration testing on **Spam** folders. If the particular contact has been marked as spam, then any mail sent by that user should go to the spam folder and not in the inbox.

As we can see in the below image, we will perform the **functional testing** for all the **text fields and every feature**. Then we will perform **integration testing** for the related functions. We first test the **add user, list of users, delete user, edit user,** and then **search user**.



Note:

- There are some features, we might be performing only the **functional testing**, and there are some features where we are performing both **functional** and **integration testing** based on the feature's requirements.
- **Prioritizing is essential,** and we should perform it at all the phases, which means we will open the application and select which feature needs to be tested first. Then go to that feature and choose which component must be tested first. Go to those components and determine what values to be entered first. And don't apply the same rule everywhere because testing logic varies from feature to feature.
- While performing testing, we should test one feature entirely and then only proceed to another function.
- Among the two features, we must be performing **only positive integrating testing** or both **positive and negative integration** testing, and this also depends on the features need.

Reason Behind Integration Testing

Although all modules of software application already tested in unit testing, errors still exist due to the following reasons:

1. Each module is designed by individual software developer whose programming logic may differ from developers of other modules so; integration testing becomes essential to determine the working of software modules.
2. To check the interaction of software modules with the database whether it is an erroneous or not.
3. Requirements can be changed or enhanced at the time of module development. These new requirements may not be tested at the level of unit testing hence integration testing becomes mandatory.
4. Incompatibility between modules of software could create errors.
5. To test hardware's compatibility with software.
6. If exception handling is inadequate between modules, it can create bugs.

Integration Testing Techniques

Any testing technique (Blackbox, Whitebox, and Greybox) can be used for Integration Testing; some are listed below:

Black Box Testing

- State Transition technique
- Decision Table Technique
- Boundary Value Analysis
- All-pairs Testing
- Cause and Effect Graph
- Equivalence Partitioning
- Error Guessing

White Box Testing

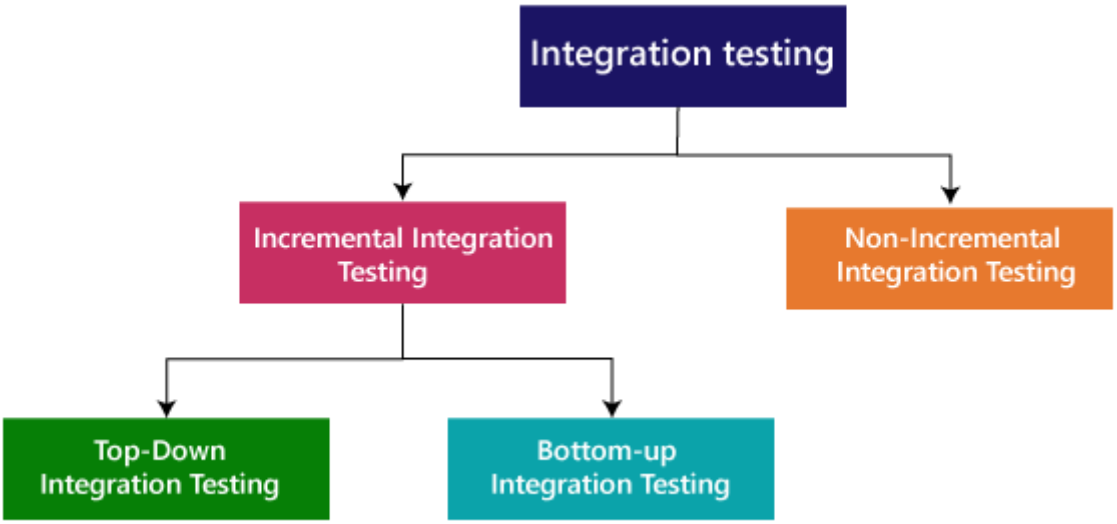
- Data flow testing
- Control Flow Testing
- Branch Coverage Testing
- Decision Coverage Testing

Types of Integration Testing

Integration testing can be classified into two parts:

- **Incremental integration testing**

- **Non-incremental integration testing**

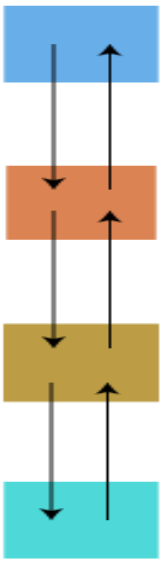


Incremental Approach

In the Incremental Approach, modules are added in ascending order one by one or according to need. The selected modules must be logically related. Generally, two or more than two modules are added and tested to determine the correctness of functions. The process continues until the successful testing of all the modules.

OR

In this type of testing, there is a strong relationship between the dependent modules. Suppose we take two or more modules and verify that the data flow between them is working fine. If it is, then add more modules and test again.



For example: Suppose we have a Flipkart application, we will perform incremental integration testing, and the flow of the application would like this:

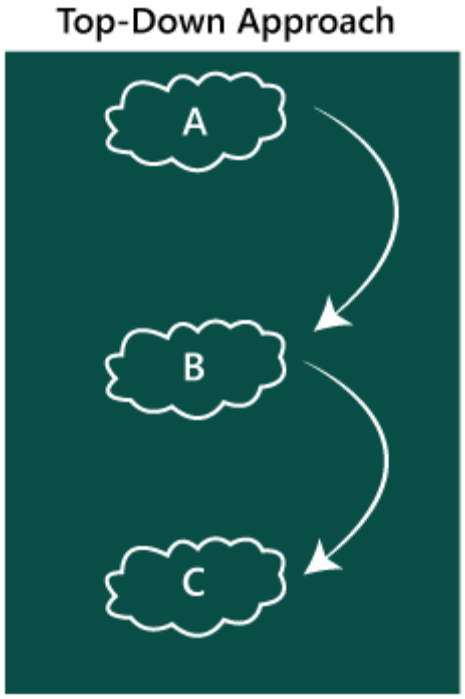
Flipkart→ Login→ Home → Search→ Add cart→Payment → Logout

Incremental integration testing is carried out by further methods:

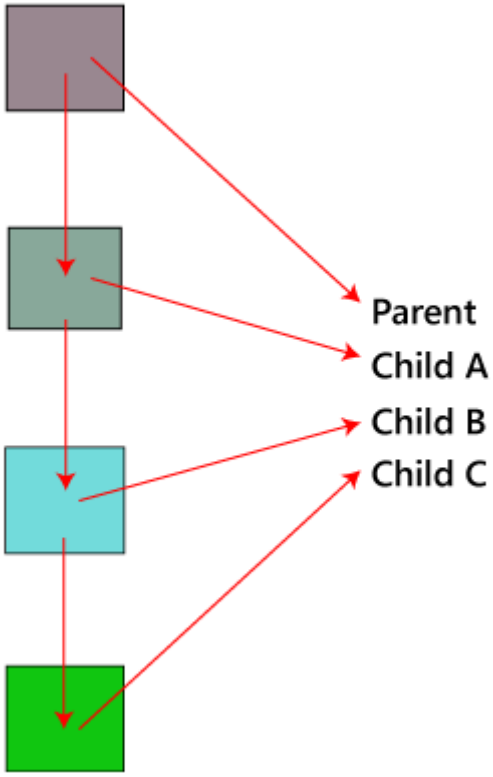
- Top-Down approach
- Bottom-Up approach

Top-Down Approach

The top-down testing strategy deals with the process in which higher level modules are tested with lower level modules until the successful completion of testing of all the modules. Major design flaws can be detected and fixed early because critical modules tested first. In this type of method, we will add the modules incrementally or one by one and check the data flow in the same order.



In the top-down approach, we will be ensuring that the module we are adding is the **child of the previous one like Child C is a child of Child B** and so on as we can see in the below image:



Advantages:

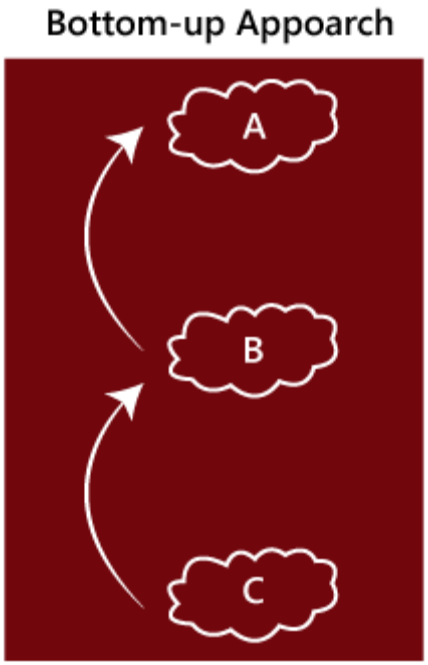
- Identification of defect is difficult.
- An early prototype is possible.

Disadvantages:

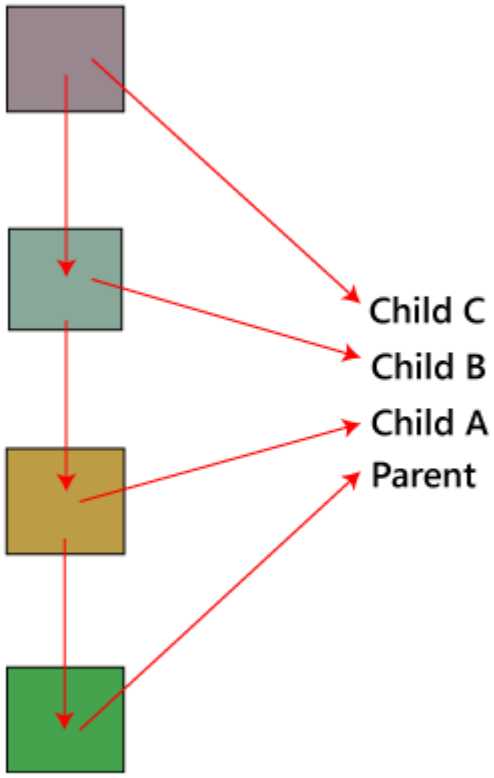
- Due to the high number of stubs, it gets quite complicated.
- Lower level modules are tested inadequately.
- Critical Modules are tested first so that fewer chances of defects.

Bottom-Up Method

The bottom to up testing strategy deals with the process in which lower level modules are tested with higher level modules until the successful completion of testing of all the modules. Top level critical modules are tested at last, so it may cause a defect. Or we can say that we will be adding the modules from **bottom to the top** and check the data flow in the same order.



In the bottom-up method, we will ensure that the modules we are adding **are the parent of the previous one** as we can see in the below image:



Advantages

- Identification of defect is easy.

- Do not need to wait for the development of all the modules as it saves time.

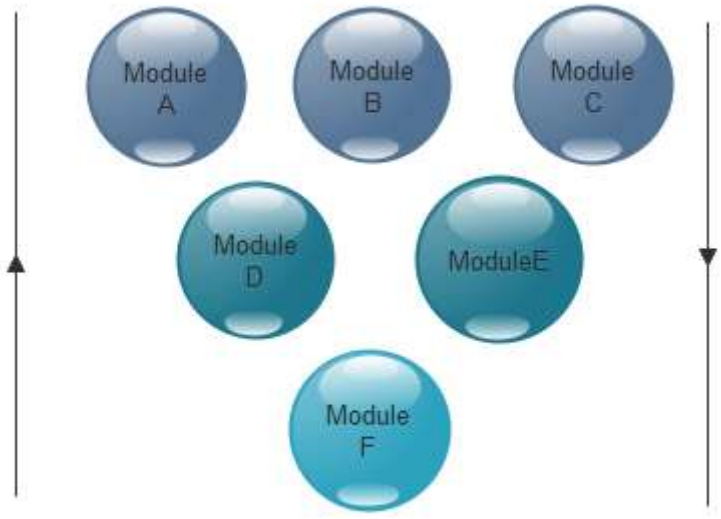
Disadvantages

- Critical modules are tested last due to which the defects can occur.
- There is no possibility of an early prototype.

In this, we have one addition approach which is known as **hybrid testing**.

Hybrid Testing Method

In this approach, both **Top-Down** and **Bottom-Up** approaches are combined for testing. In this process, top-level modules are tested with lower level modules and lower level modules tested with high-level modules simultaneously. There is less possibility of occurrence of defect because each module interface is tested.



Hybrid Method

Advantages

- The hybrid method provides features of both Bottom Up and Top Down methods.
- It is most time reducing method.
- It provides complete testing of all modules.

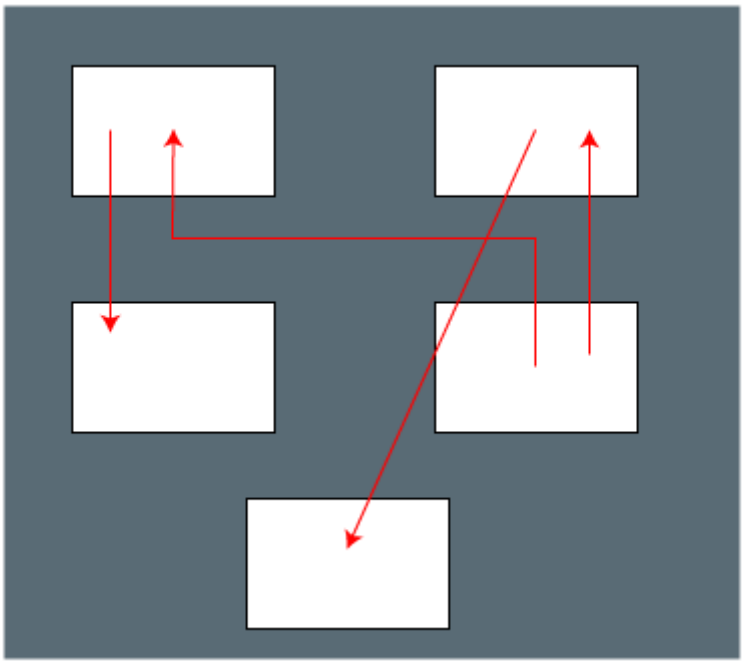
Disadvantages

- This method needs a higher level of concentration as the process carried out in both directions simultaneously.

- Complicated method.

Non- incremental integration testing

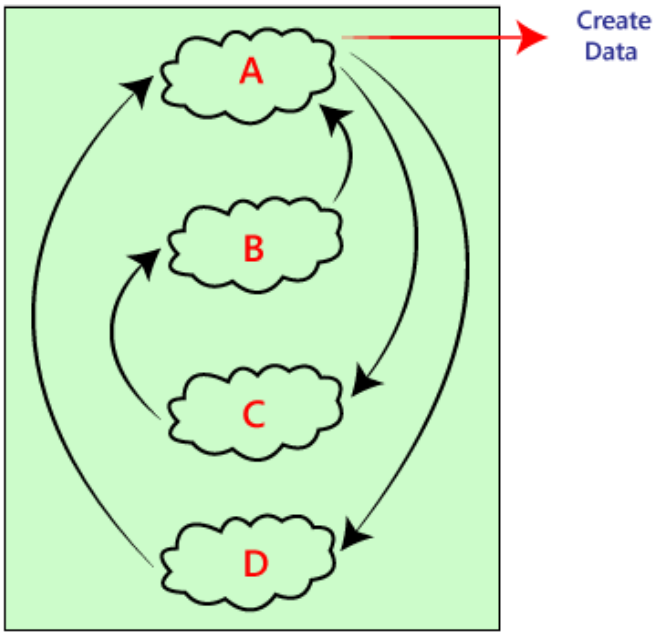
We will go for this method, when the data flow is very complex and when it is difficult to find who is a parent and who is a child. And in such case, we will create the data in any module bang on all other existing modules and check if the data is present. Hence, it is also known as the **Big bang method**.



Big Bang Method

In this approach, testing is done via integration of all modules at once. It is convenient for small software systems, if used for large software systems identification of defects is difficult.

Since this testing can be done after completion of all modules due to that testing team has less time for execution of this process so that internally linked interfaces and high-risk critical modules can be missed easily.



Advantages:

- It is convenient for small size software systems.

Disadvantages:

- Identification of defects is difficult because finding the error where it came from is a problem, and we don't know the source of the bug.
- Small modules missed easily.
- Time provided for testing is very less.
- We may miss to test some of the interfaces.

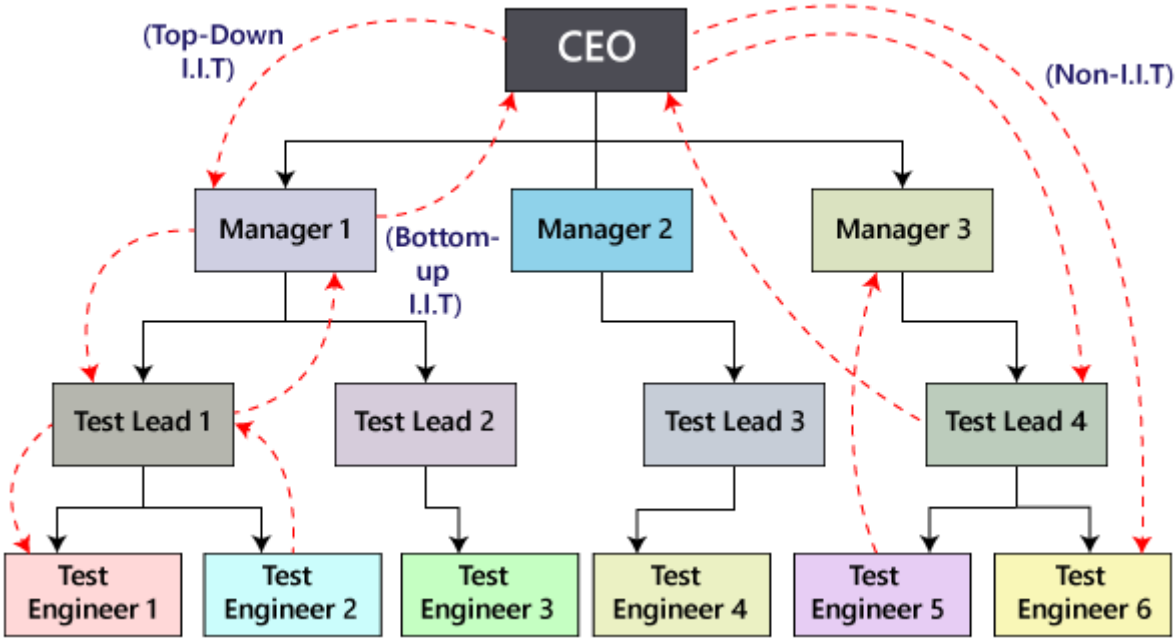
Let us see examples for our better understanding of the non-incremental integrating testing or big bang method:

Example1

In the below example, the development team develops the application and sends it to the CEO of the testing team. Then the CEO will log in to the application and generate the username and password and send a mail to the manager. After that, the CEO will tell them to start testing the application.

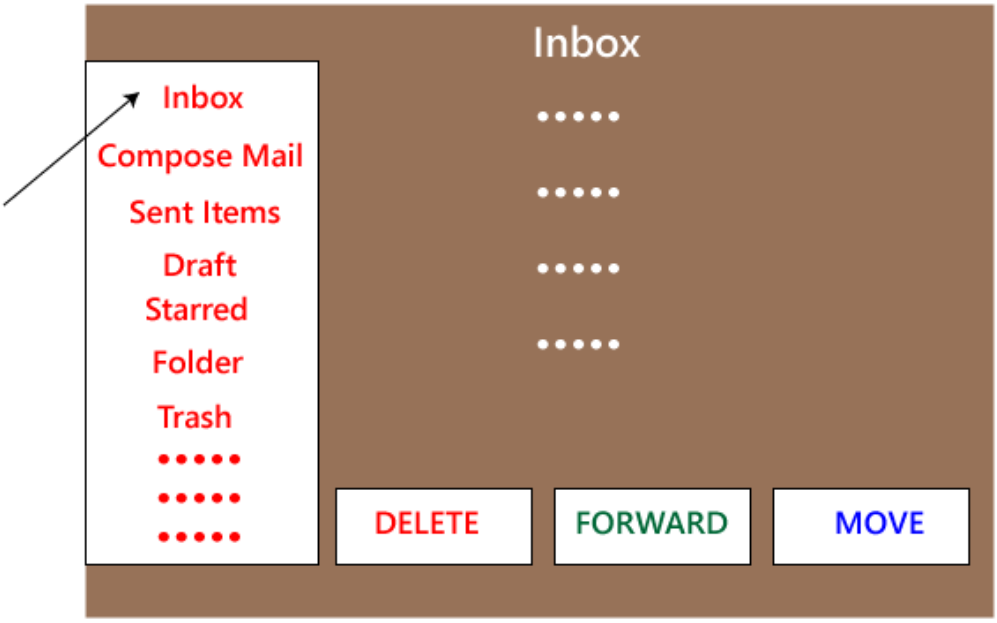
Then the manager manages the username and the password and produces a username and password and sends it to the **test leads**. And the **test leads** will send it to the **test engineers** for further testing purposes. This order from the CEO to the test engineer is **top-down incremental integrating testing.**

In the same way, when the test engineers are done with testing, they send a report to the **test leads**, who then submit a report to the **manager**, and the manager will send a report to the **CEO**. This process is known as **Bottom-up incremental integration testing** as we can see in the below image:



Example2

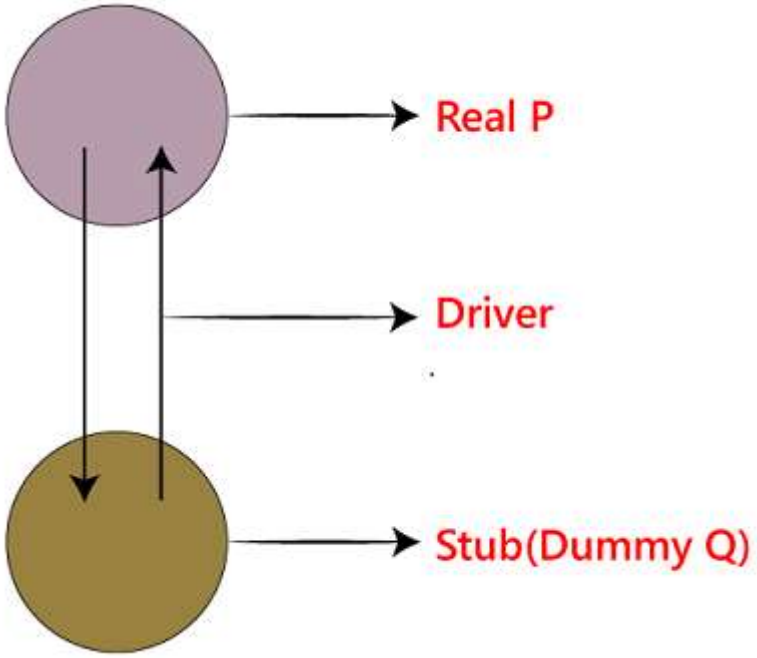
The below example demonstrates a home page of **Gmail's Inbox**, where we click on the **Inbox** link, and we are moved to the inbox page. Here we have to do **non- incremental integration testing** because there is no parent and child concept.



Note

Stub and driver

The **stub** is a dummy module that receives the data and creates lots of probable data, but it performs like a real module. When a data is sent from module P to Stub Q, it receives the data without confirming and validating it, and produce the estimated outcome for the given data.



The function of a driver is used to verify the data from P and sends it to stub and also checks the expected data from the stub and sends it to P.

The **driver** is one that sets up the test environments and also takes care of the communication, evaluates results, and sends the reports. We never use the stub and driver in the testing process.

In **White box testing**, **bottom-up integration testing** is ideal because writing drivers is accessible. And in **black box testing**, no preference is given to any testing as it depends on the application.

What is Regression Testing?

Regression testing is a black box testing technique. It is used to authenticate a code change in the software does not impact the existing functionality of the product. Regression testing is making sure that the product works fine with new functionality, bug fixes, or any change in the existing feature.

Regression testing is a type of software testing. Test cases are re-executed to check the previous functionality of the application is working fine, and the new changes have not produced any bugs.

Regression testing can be performed on a new build when there is a significant change in the original functionality. It ensures that the code still works even when the changes are occurring. Regression means Re-test those parts of the application, which are unchanged.

Regression tests are also known as the Verification Method. Test cases are often automated. Test cases are required to execute many times and running the same test case again and again manually, is time-consuming and tedious too.

Example of Regression testing

Here we are going to take a case to define the regression testing efficiently:

Consider a product Y, in which one of the functionalities is to trigger confirmation, acceptance, and dispatched emails. It also needs to be tested to ensure that the change in the code not affected them. Regressing testing does not depend on any programming language like Java, C++, C#, etc. This method is used to test the product for modifications or any updates done. It ensures that any change in a product does not affect the existing module of the product. Verify that the bugs fixed and the newly added features not created any problem in the previous working version of the Software.

When can we perform Regression Testing?

We do regression testing whenever the production code is modified.

We can perform regression testing in the following scenario, these are:

1. When new functionality added to the A website has a login functionality which allows users to log in only with email. Now providing a new feature to do login using Facebook.

2. When there is a Change Requirement.

Example:

Remember password removed from the login page which is applicable previously.

3. When the defect fixed

Example:

Assume login button is not working in a login page and a tester reports a bug stating that the login button is broken. Once the bug fixed by developers, tester tests it to make sure Login Button is working as per the expected result. Simultaneously, tester tests other functionality which is related to the login button.

4. When there is a performance issue fix

Example:

Loading of a home page takes 5 seconds, reducing the load time to 2 seconds

Example

5. When there is an environment change

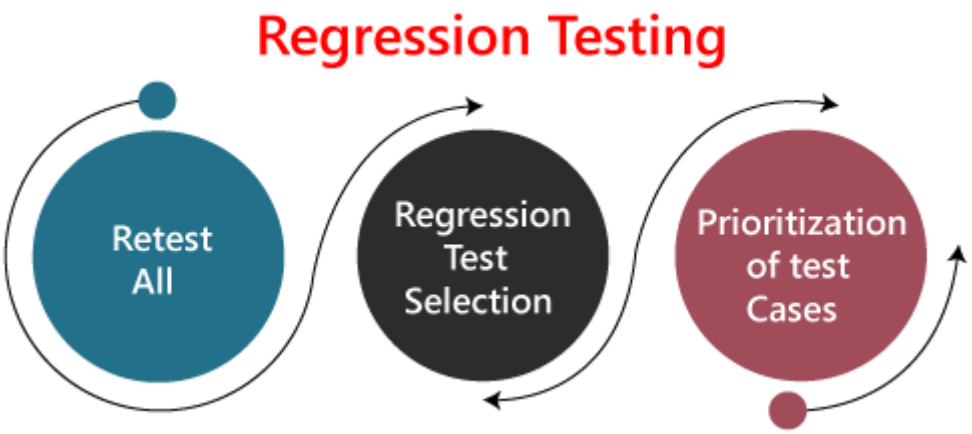
Example:

When we update the database from MySql to Oracle.

How to perform Regression Testing?

The need for regression testing comes when software maintenance includes enhancements, error corrections, optimization, and deletion of existing features. These modifications may affect system functionality. Regression Testing becomes necessary in this case.

Regression testing can be performed using the following techniques:



1. Re-test All:

Re-Test is one of the approaches to do regression testing. In this approach, all the test case suits should be re-executed. Here we can define re-test as when a test fails, and we determine the cause of the failure is a software fault. The fault is reported, we can expect a new version of the software in which defect fixed. In this case, we will need to execute the test again to confirm that the fault fixed. This is known as re-testing. Some will refer to this as confirmation testing.

The re-test is very expensive, as it requires enormous time and resources.

2. Regression test Selection:

- In this technique, a selected test-case suit will execute rather than an entire test-case suit.
- The selected test case suits divided in two cases
    1. Reusable Test cases.
    2. Obsolete Test cases.
- Reusable test cases can use in succeeding regression cycle.
- Obsolete test cases can't use in succeeding regression cycle.

3. Prioritization of test cases:

Prioritize the test case depending on business impact, critical and frequently functionality used. Selection of test cases will reduce the regression test suite.

What are the Regression Testing tools?

Regression Testing is a vital part of the QA process; while performing the regression we may face the below challenges:

- **TimeConsuming**
  Regression Testing consumes a lot of time to complete. Regression testing involves existing tests again, so testers are not excited to re-run the test.
- **Complex**
  Regression Testing is complex as well when there is a need to update any product; lists of the test are also increasing.
- **Communicating business rule**
  Regression Testing ensures the existing product features are still in working order. Communication about regression testing with a non-technical leader can be a difficult task. The executive wants to see the product move forward and making a considerable time investment in regression testing to ensure existing functionality working can be hard.
- **Identify Impact Area**
- **Test Cases Increases Release by Release**
- **Less Resources**
- **No Accuracy**
- **Repetitive Task**
- **Monotonous Job**

Regression Testing Process

The regression testing process can be performed across the **builds** and the **releases**.

Regression testing across the builds

Whenever the bug fixed, we retest the Bug, and if there is any dependent module, we go for a Regression Testing.

For example, How we perform the regression testing if we have different builds as **Build 1, Build 2, and Build 3**, which having different scenarios.

Build1

- Firstly the client will provide the business needs.
- Then the development team starts developing the features.
- After that, the testing team will start writing the test cases; for example, they write 900 test cases for the release#1 of the product.
- And then, they will start implementing the test cases.
- Once the product is released, the customer performs one round of acceptance testing.
- And in the end, the product is moved to the production server.

Build2

- Now, the customer asks for 3-4 extra (new) features to be added and also provides the requirements for the new features.
- The development team starts developing new features.
- After that, the testing team will start writing the test case for the new features, and they write about 150 new test cases. Therefore, the total number of the test case written is 1050 for both the releases.
- Now the testing team starts testing the new features using 150 new test cases.

- Once it is done, they will begin testing the old features with the help of 900 test cases to verify that adding the new feature has damaged the old features or not.
- Here, testing the old features is known as **Regression Testing**.
- Once all the features (New and Old) have been tested, the product is handed over to the customer, and then the customer will do the acceptance testing.
- Once the acceptance testing is done, the product is moved to the production server.

Build3

- After the second release, the customer wants to remove one of the features like Sales.
- Then he/she will delete all the test cases which are belonging to the sales module (about 120 test cases).
- And then, test the other feature for verifying that if all the other features are working fine after removing the sales module test cases, and this process is done under the regression testing.

Note:

- Testing the stable features to ensure that it is broken because of the changes. Here changes imply that the **modification, addition, bug fixing, or the deletion**.
- Re-execution of the same test cases in the different builds or releases is to ensure that changes (modification, addition, bug fixing, or the deletion) are not introducing bugs in stable features.

Regression Testing Across the Release

The regression testing process starts whenever there is a new Release for same project because the new feature may affect the old elements in the previous releases.

To understand the regression testing process, we will follow the below steps:

Step1

There is no regression testing in **Release#1** because there is no modification happen in the Release#1 as the release is new itself.

Step2

The concept of Regression testing starts from **Release#2** when the customer gives some **new requirements**.

Step3

After getting the new requirements (modifying features) first, they (the developers and test engineers) will understand the needs before going to the **impact analysis**.

Step4

After understanding the new requirements, we will perform one round of **impact analysis** to avoid the major risk, but here the question arises who will do the Impact analysis?

Step5

The impact analysis is done by the **customer** based on their **business knowledge**, the **developer** based on their **coding knowledge**, and most importantly, it is done by the **test engineer** because they have the **product knowledge**.

Step6

Once we are done with the **impact area**, then the developer will prepare the **impact area (document)**, and the **customer** will also prepare the **impact area document** so that we can achieve the **maximum coverage of impact analysis**.

Step7

After completing the impact analysis, the developer, the customer, and the test engineer will send the **Reports#** of the impact area documents to the **Test Lead**. And in the meantime, the test engineer and the developer are busy working on the new test case.

Step8

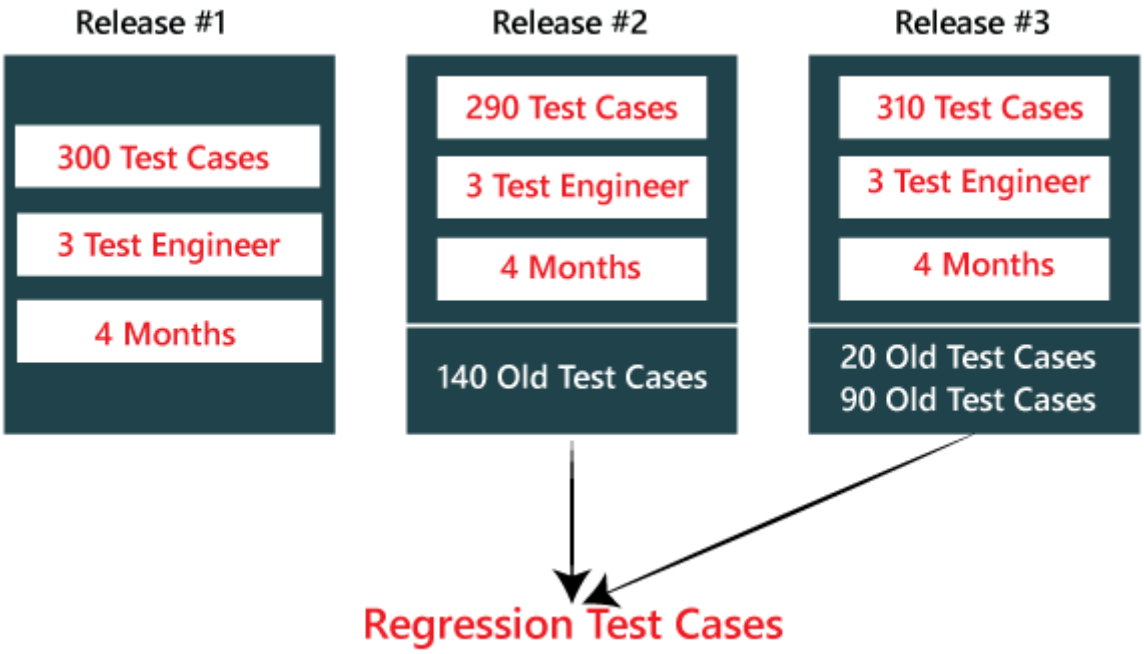
Once the Test lead gets the Reports#, he/she will **consolidate** the reports and stored in the **test case requirement repository** for the release#1.

Step9

After that, the Test Lead will take the help of RTM and pick the necessary **regression test case** from the **test case repository**, and those files will be placed in the **Regression Test Suite**.

Note:

- The test lead will store the regression test case in the regression test suite for no further confusion.
- **Regression test suite**: Here, we will save all the impact area test documents.
- **Regression Test Cases**: These are the test cases of the old releases text document which need to be re-executed as we can see in the below image:

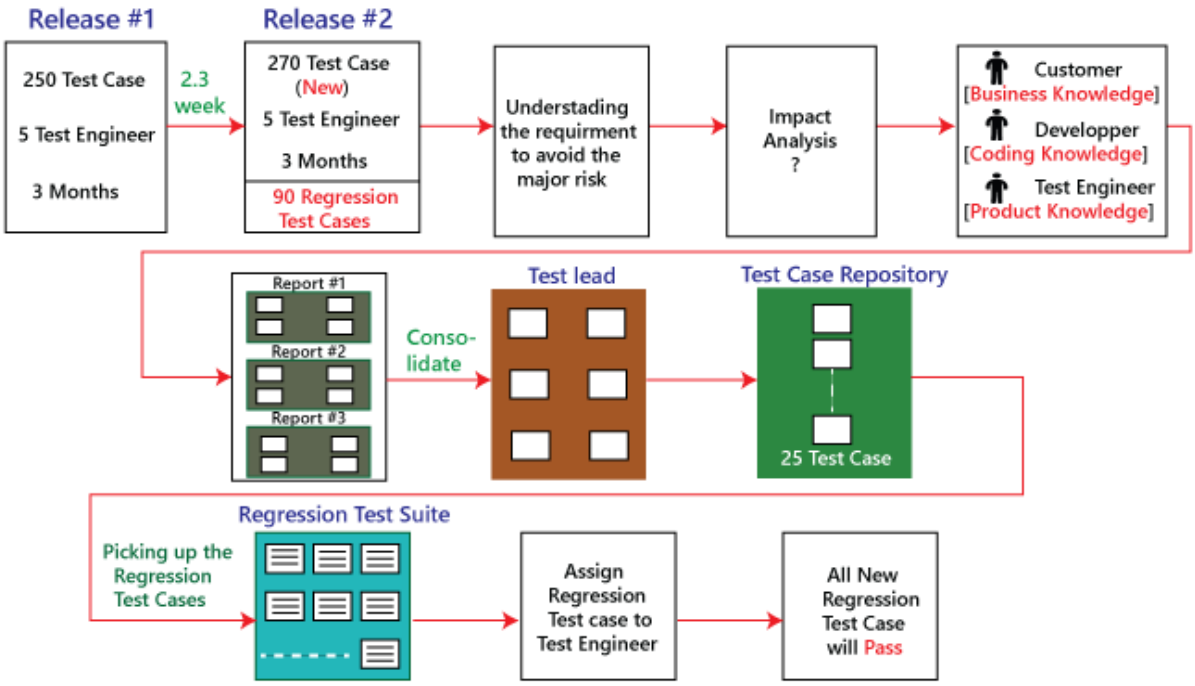


Step10

After that, when the test engineer has done working on the new test cases, the test lead will **assign the regression test case** to the test engineer.

Step11

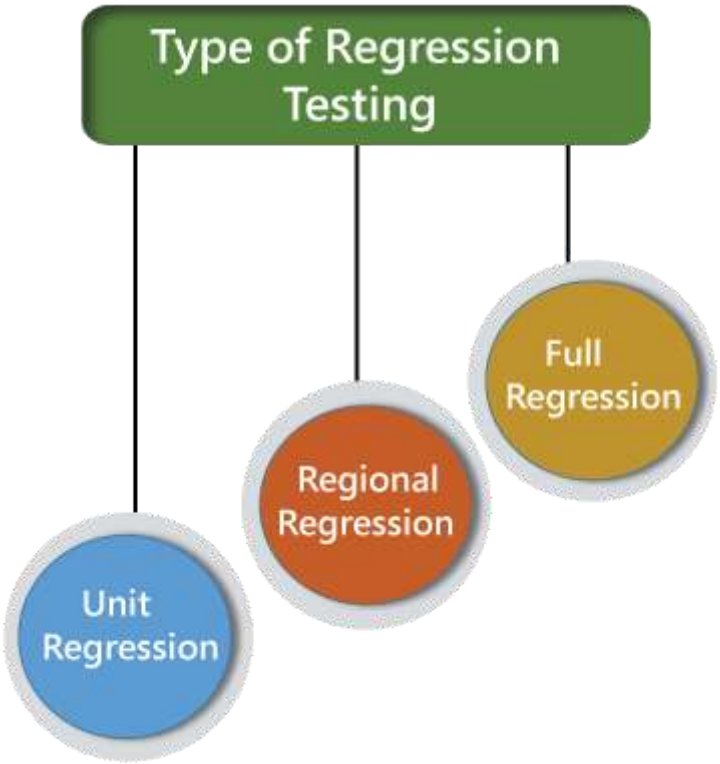
When all the regression test cases and the new features are **stable and pass**, then check the **impact area using the test case** until it is durable for old features plus the new features, and then it will be handed over to the customer.



Types of Regression Testing

The different types of Regression Testing are as follows:

1. Unit Regression Testing [URT]
2. Regional Regression Testing[RRT]
3. Full or Complete Regression Testing [FRT]

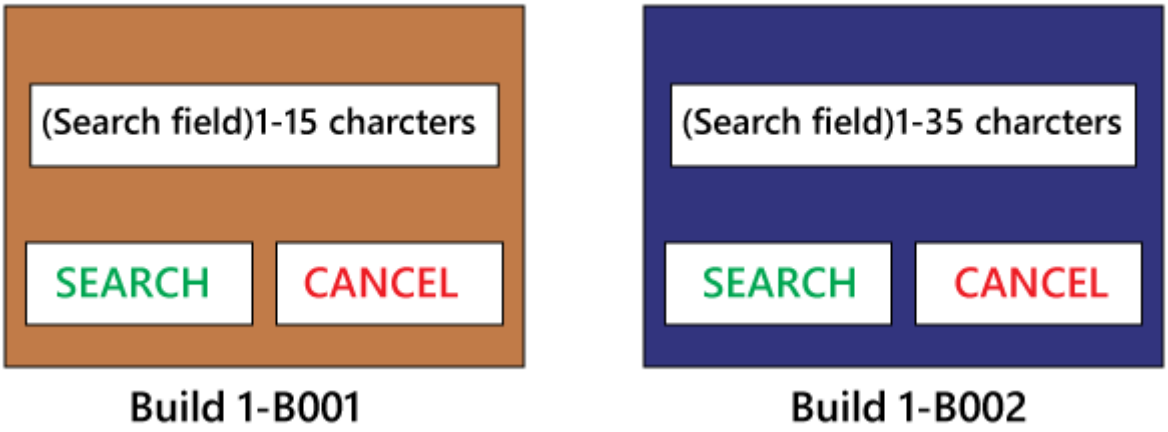


1) Unit Regression Testing [URT]

In this, we are going to test only the changed unit, not the impact area, because it may affect the components of the same module.

Example1

In the below application, and in the first build, the developer develops the **Search** button that accepts **1-15 characters**. Then the test engineer tests the Search button with the help of the **test case design technique**.



Now, the client does some modification in the requirement and also requests that the **Search button** can accept the **1-35 characters**. The test engineer will test only the Search button to verify that it takes 1-35 characters and does not check any further feature of the first build.

Example2

Here, we have **Build B001**, and a defect is identified, and the report is delivered to the developer. The developer will fix the bug and sends along with some new features which are developed in the second **Build B002**. After that, the test engineer will test only after the defect is fixed.

- The test engineer will identify that clicking on the **Submit** button goes to the blank page.
- And it is a defect, and it is sent to the developer for fixing it.
- When the new build comes along with the bug fixes, the test engineer will test only the Submit button.
- And here, we are not going to check other features of the first build and move to test the new features and sent in the second build.
- We are sure that fixing the **Submit**button is not going to affect the other features, so we test only the fixed bug.

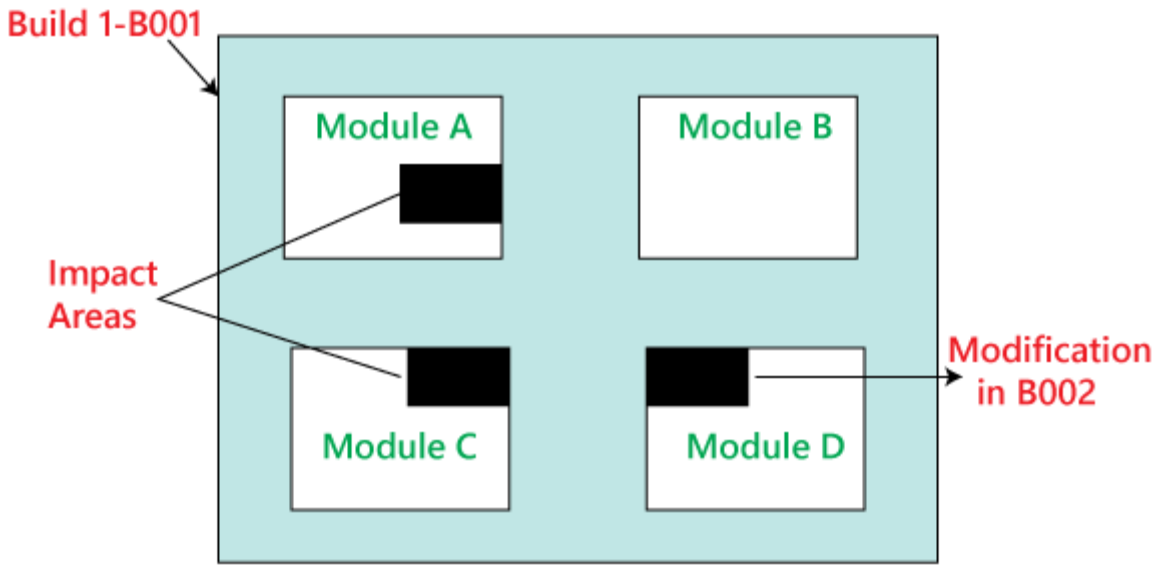
Therefore, we can say that by testing only the changed feature is called the **Unit Regression Testing**.

2) Regional Regression testing [RRT]

In this, we are going to test the modification along with the impact area or regions, are called the **Regional Regression testing**. Here, we are testing the impact area because if there are dependable modules, it will affect the other modules also.

For example:

In the below image as we can see that we have four different modules, such as **Module A, Module B, Module C, and Module D**, which are provided by the developers for the testing during the first build. Now, the test engineer will identify the bugs in **Module D**. The bug report is sent to the developers, and the development team fixes those defects and sends the second build.



In the second build, the previous defects are fixed. Now the test engineer understands that the bug fixing in Module D has impacted some features in **Module A and Module C**. Hence, the test engineer first tests the Module D where the bug has been fixed and then checks the impact areas in **Module A and Module C**. Therefore, this testing is known as **Regional regression testing.**

While performing the regional regression testing, we may face the below problem:

Problem:

In the first build, the client sends some modification in requirement and also wants to add new features in the product. The needs are sent to both the teams, i.e., development and testing.

After getting the requirements, the development team starts doing the modification and also develops the new features based on the needs.

Now, the test lead sends mail to the clients and asks them that all are the impact areas that will be affected after the necessary modification have been done. Therefore, the customer will get an idea, which all features are needed to be tested again. And he/she will also send a mail to the development team to know which all areas in the application will be affected as a result of the changes and additions of new features.

And similarly, the customer sends a mail to the testing team for a list of impact areas. Hence, the test lead will collect the impact list from the client, development team, and the testing team as well.

This **Impact list** is sent to all the test engineers who look at the list and check if their features are modified and if yes, then they do **regional regression testing**. The impact areas and modified areas are all tested by the respective engineers. Every test engineer tests only their features that could have been affected as a result of the modification.

The problem with this above approach is that the test lead may not get the whole idea of the impact areas because the development team and the client may not have so much time to revert his/her mails.

Solution

To resolve the above problem, we will follow the below process:

When a new build comes along with the latest features and bug fixes, the testing team will arrange the meeting where they will talk about if their features are affecting because of the above modification. Therefore, they will do one round of **Impact Analysis** and generate the **Impact List**. In this particular list, the test engineer tries to enclose the maximum probable impact areas, which also decreases the chance of getting the defects.

When a new build comes, the testing team will follow the below procedure:

- They will do smoke testing to check the basic functionality of an application.
- Then they will test new features.
- After that, they will check the changed features.
- Once they are done with checking the changed features, the test engineer will re-test the bugs.
- And then they will check the impact area by performing the regional regression testing.

Disadvantage of using Unit and Regional Regression testing

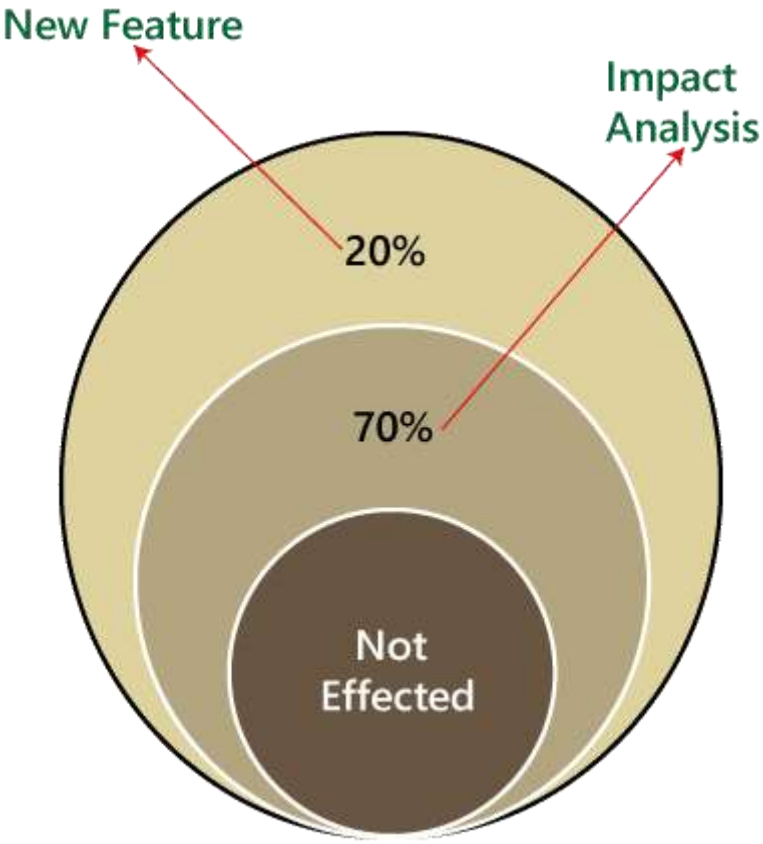
Following are some of the drawbacks of using unit and Regional regression testing:

- We may miss some impact area.
- It is possible that we may identify the wrong impact area.

3) Full Regression testing [FRT]

During the second and the third release of the product, the client asks for adding 3-4 new features, and also some defects need to be fixed from the previous release. Then the testing team will do the Impact Analysis and identify that the above modification will lead us to test the entire product.

Therefore, we can say that testing the **modified features** and **all the remaining (old) features** is called the **Full Regression testing**.



When we perform Full Regression testing?

We will perform the FRT when we have the following conditions:

- When the modification is happening in the source file of the product. **For example**, JVM is the root file of the JAVA application, and if any change is going to happen in JVM, then the entire JAVA program will be tested.
- When we have to perform n-number of changes.

Note:

The regional regression testing is the ideal approach of regression testing, but the issue is, we may miss lots of defects while performing the Regional Regression testing.

And here we are going to solve this issue with the help of the following approach:

- When the application is given for the testing, the test engineer will test the first 10-14 cycle, and will do the **RRT**.
- Then for the 15th cycle, we do FRT. And again, for the next 10-15 cycle, we do **Regional regression testing**, and for the 31th cycle, we do the **full regression testing**, and we will continue like this.

- But for the last ten cycle of the release, we will perform only **complete regression testing**.

Therefore, if we follow the above approach, we can get more defects.

The drawback of doing regression testing manually repeatedly:

- Productivity will decrease.
- It is a difficult job to do.
- There is no consistency in test execution.
- And the test execution time is also increased.

Hence, we will go for the automation to get over with these issues; when we have n-number of the regression test cycle, we will go for the **automation regression testing process**.

Automated Regression Testing Process

Generally, we go for the automation whenever there are multiple releases or multiple regression cycle or there is the repetitive task.

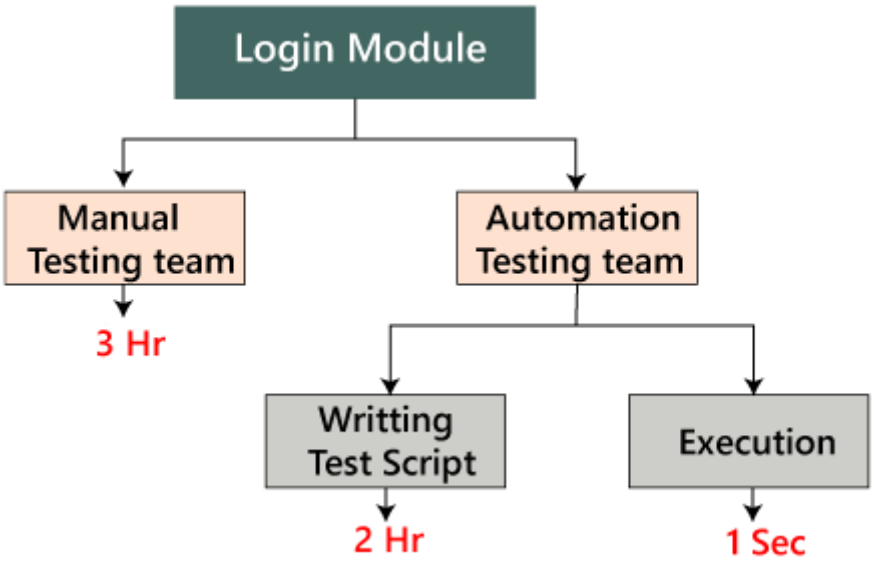
The automation regression testing process can be done in the following steps:

Note1:

The process of testing the application by using some tools is known as automation testing.

Suppose if we take one sample example of a **Login module**, then how we can perform the regression testing.

Here, the Login can be done in two ways, which are as follows:



Manually: In this, we will perform regression only one and twice.

Automation: In this, we will do the automation multiple times as we have to write the test scripts and do the execution.

Issues	Handle by
New features	Manual test engineer
Regressing testing features	Automation test engineer
Remaining (110 feature + Release#1)	Manual test engineer

Step1

When the new release starts, we don't go for the automation because there is no concept of regression testing and regression test case as we understood this in the above process.

Step2

When the new release and the enhancement starts, we have two teams, i.e., manual team and the automation team.

Step3

The manual team will go through the requirements and also identify the impact area and hand over the **requirement test suite** to the automation team.

Step4

Now, the manual team starts working on the new features, and the automation team will start developing the test script and also start automating the test case, which means that the regression test cases will be converted into the test script.

Step5

Before they (automation team) start automating the test case, they will also analyze which all cases can be automated or not.

Step6

Based on the analysis, they will start the automation i.e., converting every regression test cases into the test script.

Step7

During this process, they will take help of the **Regression cases** because they don't have product knowledge as well as the **tool** and the **application**.

Step8

Once the test script is ready, they will start the execution of these scripts on the new application [old feature]. Since, the test script is written with the help of the regression feature or the old feature.

Step9

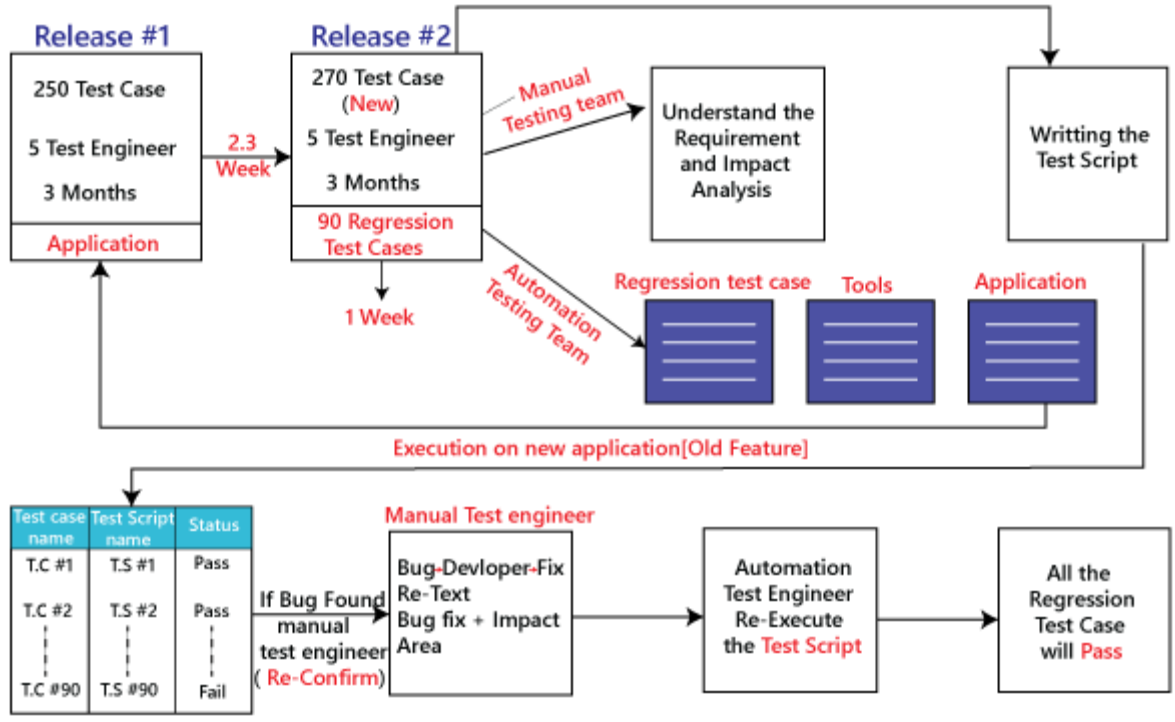
Once the execution is completed, we get a different status like **Pass/fail**.

Step10

If the status is failed, which means it needs to be re-confirmed manually, and if the Bug exists, then it will report to the concerned developer. When the developer fixes that bug, the Bug needs to be re-tested along with the Impact area by the manual test engineer, and also the script needs to be re-executed by the automation test engineer.

Step11

This process goes on until all the new features, and the regression feature will be passed.



Benefits of doing regression testing by the automation testing:

- **Accuracy** always exists because the task is done by the tools and tools never get bored or tired.

- The test script can be re-used across multiple releases.
- **Batch execution** is possible using the automation i.e.; all the written test scripts can be executed parallel or simultaneously.
- Even though the number of regression test case increase release per release, and we don't have to increase the automation resource since some regression case are already automated from the previous release.
- It is a **time-saving process** because the execution is always faster than the manual method.

How to select test cases for regression testing?

It was found from industry inspection. The several defects reported by the customer were due to last-minute bug fixes. These creating side effects and hence selecting the Test Case for regression testing is an art, not an easy task.

Regression test can be done by:

- A test case which has frequent defects
- Functionalities which are more visible to users.
- Test cases verify the core features of the product.
- All integration test cases
- All complex test cases
- Boundary value test cases
- A sample of successful test cases
- Failure of test cases

Regression Testing Tools

If Software undergoes frequent changes, regression testing costs also increase. In those cases, manual execution of test cases increases test execution time as well as costs. In that case, automation testing is the best choice. The duration of automation depends on the number of test cases that remain reusable for successive regression cycles.

Following are the essential tools used for regression testing:

Selenium

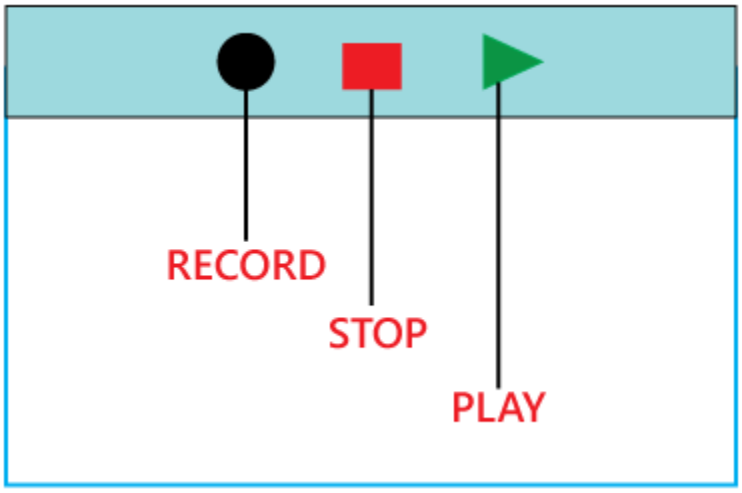
Selenium is an open-source tool. This tool used for automated testing of a web application. For browser-based regression testing, selenium used. Selenium used for UI level regression test for web-based application.

Ranorex Studio

All in one regression test automation for desktop, web, and mobile apps with built-in Selenium Web Driver. Ranorex Studio includes full IDE plus tools for codeless automation.

Quick Test Professional (QTP)

QTP is an automated testing tool used for Regression and Functional Testing. It is a Data-Driven, keyword-based tool. It used VBScript language for automation. If we open the QTP tool, we see the three buttons which are **Record, Play and Stop**. These buttons help to record every click and action performed on the computer system. It records the actions and play it back.



Rational Functional Tester (RTF)

Rational functional tester is a Java tool used to automate the test cases of software applications. RTF used for automating regression test cases, and it also integrates with the rational functional tester.

For more information about regression and automation testing tools refer to the below link:

Regression Testing and Configuration Management

Configuration Management in the regression testing becomes imperative in Agile Environments, where a code is continuously modified. To ensure a valid regression test, we must follow the steps:

- Changes are not allowed in the code during the regression testing phase.
- A regression test case must be unaffected developer changes.
- The database used for regression testing must be isolated; changes are not allowed in the database.

Differences between Retesting and Regression Testing

Re-testing Testing means testing the functionality or bug again to ensure the code fixed. If not set, defects need not be re-opened. If fixed, the defect closed.

Re-testing is a type of testing which performed to check the test-cases that were unsuccessful in the final execution are successfully pass after the defects repaired.

Regression Testing means testing the software application when it undergoes a code change to ensure that new code has not affected other parts of the Software.

Regression testing is a type of testing executed to check whether a code has not changed the existing functionality of the application.

Re-testing	Regression Testing
Re-testing is performed to ensure that the test cases that are failed in the final execution are passing after the defects fixed.	Regression Testing is done to confirm whether the code change has not affected the existing features.
Re-Testing works on defect fixes.	The purpose of regression testing is to ensure that the code changes adversely not affect the existing functionality.
Defect verification is the part of the Retesting.	Regression testing does not include defect verification
The priority of Retesting is higher than Regression Testing, so it is done before the Regression Testing.	Based on the project type and availability of resources, regression testing can be parallel to Retesting.
Re-Test is a planned Testing.	Regression testing is a generic Testing.
We cannot automate the test-cases for Retesting.	We can do automation for regression testing; manual testing could be expensive and time-consuming.
Re-testing is for failed test-cases.	Regression testing is for passed Test-cases.
Re-testing make sure that the original fault is corrected.	Regression testing checks for unexpected side effect.

Retesting executes defects with the same data and the same environment with different input with a new build.	Regression testing is when there is a modification or changes become mandatory in an existing project.
Re-testing cannot do before start testing.	Regression testing can obtain test cases from the functional specification, user tutorials and manuals, and defects reports in regards to the corrected problem.

Differences between the Re-testing and Regression Testing are as follows:

Advantages of Regression Testing

Advantages of Regression Testing are:

- Regression Testing increases the product's quality.
- It ensures that any bug fix or changes do not impact the existing functionality of the product.
- Automation tools can be used for regression testing.
- It makes sure the issues fixed do not occur again.

Disadvantages of Regression Testing

There are several advantages of Regression Testing though there are disadvantages as well.

- Regression Testing should be done for small changes in the code because even a slight change in the code can create issues in the existing functionality.
- If in case automation is not used in the project for testing, it will time consuming and tedious task to execute the test again and again.

Performance Testing vs Functional Testing

Testing is an important phase in the software development life cycle. It checks and validates all kinds of programs to determine malfunctions that could introduce trouble later or in the future such as processing issues, usability, and so on. These problems are very dangerous from a business point of view that profit by selling software. When a company loses its customer trust, it loses reputation, loyalty, revenue, and so on. Functional testing and performance testing are one of the two testing and QA that entitle developers and performance engineers to guarantee the quality of code.

Performance Testing

Performance testing is a type of software testing that ensures software applications perform as expected under load. It is a testing technique used to determine system performance in terms of sensitivity, reactivity, and stability under specific workload conditions.

Performance testing is a type of software testing that focuses on evaluating a system's or application's performance and scalability. Performance testing aims to identify bottlenecks, measure system performance under varying loads and conditions, and ensure that the system can handle the expected number of users or transactions.

Functional Testing

Functional testing is a type of testing that ensures that each function of a software application operates to the requirements and specifications. This testing is not concerned with the application's source code. Each software application functionality is tested by providing appropriate test input, anticipating the output, and comparing the actual output to the expected output. This testing focuses on the Application Under Test's user interface, APIs, database, security, client or server application, and functionality. Manual or automated functional testing is available.

Performance Testing vs Functional Testing

Difference between Performance testing and Functional testing:

Aspect Functional Testing Performance Testing

Purpose Verifies that software functions as intended and meets specified requirements. Evaluates the system's performance under various conditions like load, stress, and scalability.

Focus Tests individual functions or features to ensure correct behavior. Measures responsiveness, speed, and stability of the entire system.

Scope Includes unit testing, integration testing, system testing, etc.Involves load testing, stress testing, scalability testing, etc.

Testing Criteria Validates functionality, user interface, data handling, etc. Assesses speed, reliability, scalability, and resource usage.

Examples Unit testing, integration testing, system testing, acceptance testing, etc. Load testing, stress testing, endurance testing, scalability testing, etc.

Key Metrics Pass/fail based on functional requirements. Response time, throughput, resource utilization, error rates, etc.

Users' Perspective Concerned with what the system does. Concerned with how well the system performs under different conditions.

Tools Selenium, JUnit, TestNG, etc. Apache JMeter, LoadRunner, Gatling, etc.

Conclusion:

Performance testing ensures that application software can handle real-world scenarios and address any issues that may arise in order to deliver a robust and efficient product to end users. Functional testing, on the other hand, ensures that software is valid in terms of functional and business requirements.

Top-Down and Bottom-Up

In **System Design**, there are two types of approaches followed namely, the **Bottom-Up Model** and the **Top-Down Model**.

- The bottom-up model is one in which the different parts of a system are designed and developed and then all these parts are connected together as a single unit.
- On the other hand, the top-down model is one in which the whole system is decomposed into smaller sub-components, then each of these parts are designed and developed till the completed system is designed.

Read this article to find out more about the bottom-up model and the top-down model of system design and how they are different from each other.

What is Bottom-Up Model?

Bottom-Up Model is a system design approach where the parts of a system are defined in details. Once these parts are designed and developed, then these parts or components are linked together to prepare a bigger component. This approach is repeated until the complete system is built.

The advantage of Bottom-Up Model is in making decisions at very low level and to decide the re-usability of components.

What is Top-Down Model?

Top-Down Model is a system design approach where the design starts from the system as a whole. The complete system is then divided into smaller sub-applications with more details.

Each part again goes through the top-down approach till the complete system is designed with all the minute details. Top-Down approach is also termed as breaking a bigger problem into smaller problems and solving them individually in recursive manner.

Difference between Bottom-Up Model and Top-Down Model

The following are the important differences between Bottom-Up Model and Top-Down Model −

Key	Bottom-Up Model	Top-Down Model
Focus	In Bottom-Up Model, the focus is on identifying and resolving smallest problems and then integrating them together to solve the bigger problem.	In Top-down Model, the focus is on breaking the bigger problem into smaller one and then repeat the process with each problem.
Language	Bottom-Up Model is mainly used by object oriented programming languages like **Java**, **C++**, etc.	Top-Down Model is followed by structural programming languages like **C**, **Fortran**, etc.
Redundancy	Bottom-Up model is better suited as it ensures minimum data redundancy and focus is on re-usability.	Top-down model has high ratio of redundancy as the size of project increases.
Interaction	Bottom-Up model have high interactivity between various modules.	Top-down model has tight coupling issues and low interactivity between various modules.
Approach	Bottom-up model is based on composition approach.	Top-down model is based on decomposition approach.
Issues	In Bottom-Up, sometimes it is difficult to identify overall functionality of system in initial stages.	In Top-Down, it may not be possible to break the problem into set of smaller problems.

Conclusion

The most significant difference between the two types of models is that the bottom-up model is based on the composition approach, while the top-down model is based on the decomposition approach.

Another important difference between the two is that the top-down model is mainly used in structural programming like C programming, whereas the bottom-up approach is followed in **object-oriented programming** like C++, Java, etc.

Acceptance Testing

Acceptance Testing – Software Testing

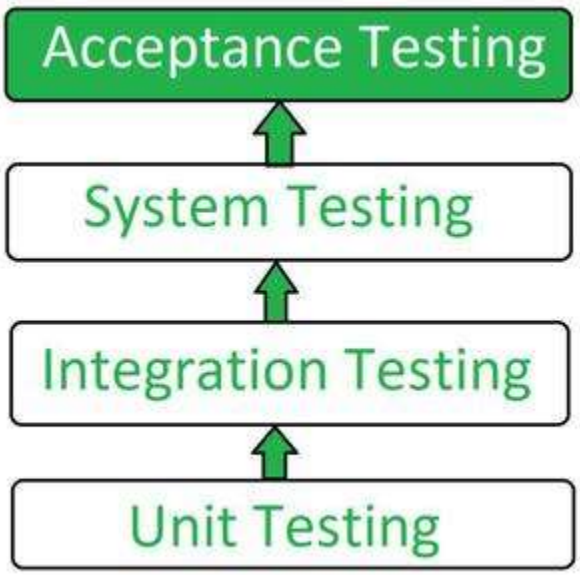
Acceptance Testing is a method of software testing where a system is tested for acceptability. The major aim of this test is to evaluate the compliance of the system with the business requirements and assess whether it is acceptable for delivery or not.

Standard Definition of Acceptance Testing

It is formal testing according to user needs, requirements, and business processes conducted to determine whether a system satisfies the acceptance criteria or not and to enable the users, customers, or other authorized entities to determine whether to accept the system or not.

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

Types of Acceptance Testing



1. User Acceptance Testing (UAT)

User acceptance testing is used to determine whether the product is working for the user correctly. Specific requirements which are quite often used by the customers are primarily picked for testing purposes. This is also termed as *End-User* Testing.

2. Business Acceptance Testing (BAT)

BAT is used to determine whether the product meets the business goals and purposes or not. BAT mainly focuses on business profits which are quite challenging due to the changing market conditions and new technologies, so the current implementation may have to being changed which results in extra budgets.

3. Contract Acceptance Testing (CAT)

CAT is a contract that specifies that once the product goes live, within a predetermined period, the acceptance test must be performed, and it should pass all the acceptance use cases. Here is a contract termed a Service Level Agreement (SLA), which includes the terms where the payment will be made only if the Product services are in-line with all the requirements, which means the contract is fulfilled. Sometimes, this contract happens before the product goes live. There should be a well-defined contract in terms of the period of testing, areas of testing, conditions on issues encountered at later stages, payments, etc.

4. Regulations Acceptance Testing (RAT)

RAT is used to determine whether the product violates the rules and regulations that are defined by the government of the country where it is being released. This may be unintentional but will impact negatively on the business. Generally, the product or application that is to be released in the market, has to go under RAT, as different countries or regions have different rules and regulations defined by its governing bodies. If any rules and regulations are violated for any country then that country or the specific region then the product will not be released in that country or region. If the product is released even though there is a violation then only the vendors of the product will be directly responsible.

5. Operational Acceptance Testing (OAT)

OAT is used to determine the operational readiness of the product and is non-functional testing. It mainly includes testing of recovery, compatibility, maintainability, reliability, etc. OAT assures the stability of the product before it is released to production.

6. Alpha Testing

Alpha testing is used to determine the product in the development testing environment by a specialized testers team usually called alpha testers.

7. Beta Testing

Beta testing is used to assess the product by exposing it to the real end-users, typically called beta testers in their environment. Feedback is collected from the users and the defects are fixed. Also, this helps in enhancing the product to give a rich user experience.

Use of Acceptance Testing

1. To find the defects missed during the functional testing phase.
2. How well the product is developed.
3. A product is what actually the customers need.
4. Feedback help in improving the product performance and user experience.
5. Minimize or eliminate the issues arising from the production.

Advantages of Acceptance Testing

1. This testing helps the project team to know the further requirements from the users directly as it involves the users for testing.
2. Automated test execution.
3. It brings confidence and satisfaction to the clients as they are directly involved in the testing process.
4. It is easier for the user to describe their requirement.
5. It covers only the Black-Box testing process and hence the entire functionality of the product will be tested.

Disadvantages of Acceptance Testing

1. Users should have basic knowledge about the product or application.
2. Sometimes, users don’t want to participate in the testing process.
3. The feedback for the testing takes a long time as it involves many users and the opinions may differ from one user to another user.
4. Development team is not participated in this testing process.