
SR
UNIVERSITY

Software Testing Methodologies

Term: 2023-24

Unit-1

Text Books: 1. Software Engineering, A practitioner's approach
Roger s. Pressman 6th edition McGraw-Hill
2. Software Engineering Somerville 7th edition

Unit-1 Syllabus

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.

What is Software?

Software is: (1) instructions (computer programs) that when executed provide desired features, function, and performance; (2) data structures that enable the programs to adequately manipulate information and (3) documentation that describes the operation and use of the programs.

Software = Instructions + Data Structures + Documentations

What is Software?

- *Software is developed or engineered, it is not manufactured in the classical sense.*
- *Software doesn't "wear out."*
- *Although the industry is moving toward component-based construction, most software continues to be custom-built.*

The Evolving role of software

Dual role of Software

A Product

- *Information transformer-*
  producing, managing and displaying

A Vehicle for delivering a product

- *Control of computer(operating system),the communication of information(networks) and the creation of other programs*

Definition of Engineering

- *-Application of science, tools and methods to find cost effective solution to problems*

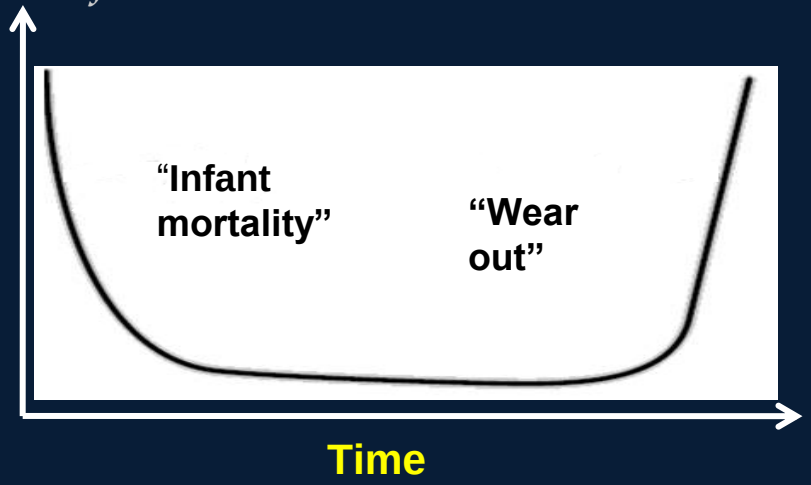
Definition of SOFTWARE ENGINEERING

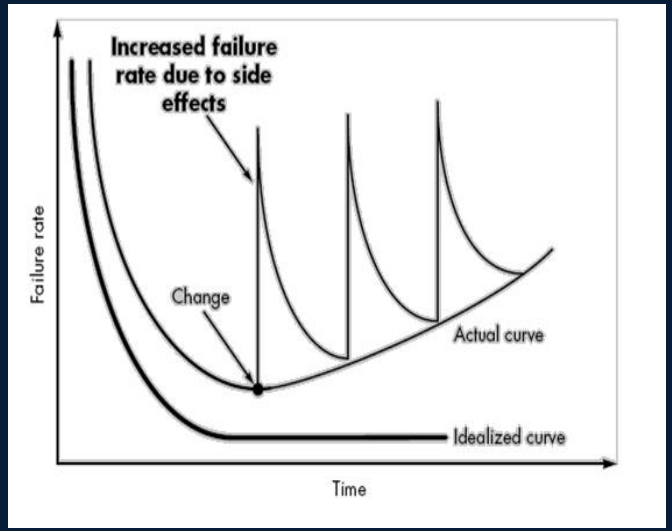
- *- SE is defined as systematic, disciplined and quantifiable approach for the development, operation and maintenance of software*

Characteristics of Software

- *Software is developed or engineered, it is not manufactured in the classical sense.*
- *Software does not wear out. However 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 OF SOFTWARE

Seven Broad Categories of software are challenges for software engineers

- *System software*
- *Application software*
- *Engineering and scientific software*
- *Embedded software*
- *Product-line software*
- *Web-applications*
- *Artificial intelligence software*

- ***System software:*** *System software is a collection of programs written to service other programs. System software: such as compilers, editors, file management utilities.*
- ***Application Software:*** *stand-alone programs for specific needs. This software are used to controls business needs. Ex: Transaction processing, MS Office.*
- ***Embedded software:*** *This software resides within a system or product and it is used to implement and control features and functions from end user and system itself. This software performs limited function like keypad control for microwave oven.*
- ***Artificial intelligence software:*** *Artificial intelligence (AI) software makes use of nonnumeric algorithms to solve complex problems. Application within this area include robotics, pattern recognition, game playing.*
- ***Engineering and scientific software:*** *Engineering and scientific software have been characterized by "number crunching" algorithm.*

- ***Product-line software:*** *focus on a limited marketplace to address mass consumer market. (word processing, graphics, database management)*

- ***WebApps (Web applications):*** *network centric software. As web 2.0 emerges, more sophisticated computing environments is supported integrated with remote database and business applications.*

LEGACY SOFTWARE

Legacy software is older programs that were developed decades ago. The quality of legacy software is poor because it has an inextensible design, convoluted code, poor and nonexistent documentation, test cases, and results that are not achieved.
As time passes, legacy systems evolve due to the following reasons:

- The software must be adapted to meet the needs of the 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 databases.
- The software must be rearchitected to make it viable within a network environment

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.

SOFTWARE MYTHS

Widely held but false view

Propagate misinformation and confusion

Three types of myths

- *Management myth*
- *Customer myth*
- *Practitioner's myth*

Management Myths

Myth(1)

- The available standards and procedures for software are enough.

Myth(2)

-Each organization feel that they have state-of-art software development tools since they have latest computer.

Myth(3)

-Adding more programmers when the work is behind schedule can catch up.

Myth(4)

-Outsourcing the software project to third party, we can relax and let that party build it.

Customer Myths

Myth(1)

- General statement of objective is enough to begin writing programs, the details can be filled in later.

Myth(2)

-Software is easy to change because software is flexible

PRACTITIONER'S MYTH

Myth(1)

-Once the program is written, the job has been done.

Myth(2)

-Until the program is running, there is no way of assessing the quality.

Myth(3)

-The only deliverable work product is the working program

Myth(4)

-Software Engineering creates voluminous and unnecessary documentation and invariably slows down software development.

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.*

Similarities

- *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.*

Differences

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.

Testing is an integral part of the development process.	Testing occurs mainly after product completion.
Prototyping is common and helps to refine requirements.	Prototyping is less common due to cost and time.
Maintenance and updates are necessary to keep software relevant.	Maintenance is typically scheduled or reactive.
Software development often involves complex logic and algorithms.	Conventional engineering may have more complex physical properties to deal with.
Software development often follows established standards and frameworks.	Conventional engineering may have well-established regulations and standards.

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.

- **Architecture:** It is a key property of software system. It describes the internal structure of software system. A good, simple structure makes the system easy to understand, change, test and maintain.
- **Correctness:** A software product satisfies given needs. Also it is free from error (reliable).
- **Ease of use:** It should be easy to learn and use.
- **Efficiency:** It is the ability to produce high performance software with optimum use of available hardware and software resources.
- **Extensibility:** With minimum effect to accommodate changes in the requirement, design decisions or due to change in technical constraints.

- **On Time:** Software should be developed well in time.
- **Portability:** It is the ease with which software is transformed to other hardware and or software platforms.
- **Readability:** It is the ability to produce a bug free software.
- **Reusability:** It is the ability so that the whole software or a part of the software can be reused in other software.

- **Robustness:** It ensures whether the software can continue its process in spite of unexpected problems.
- **Usability:** Software should have an appropriate user interaction and meet the need of intended users.
- **Within Budget:** The software development costs should not overrun and it should be within the budgetary limits.

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.*

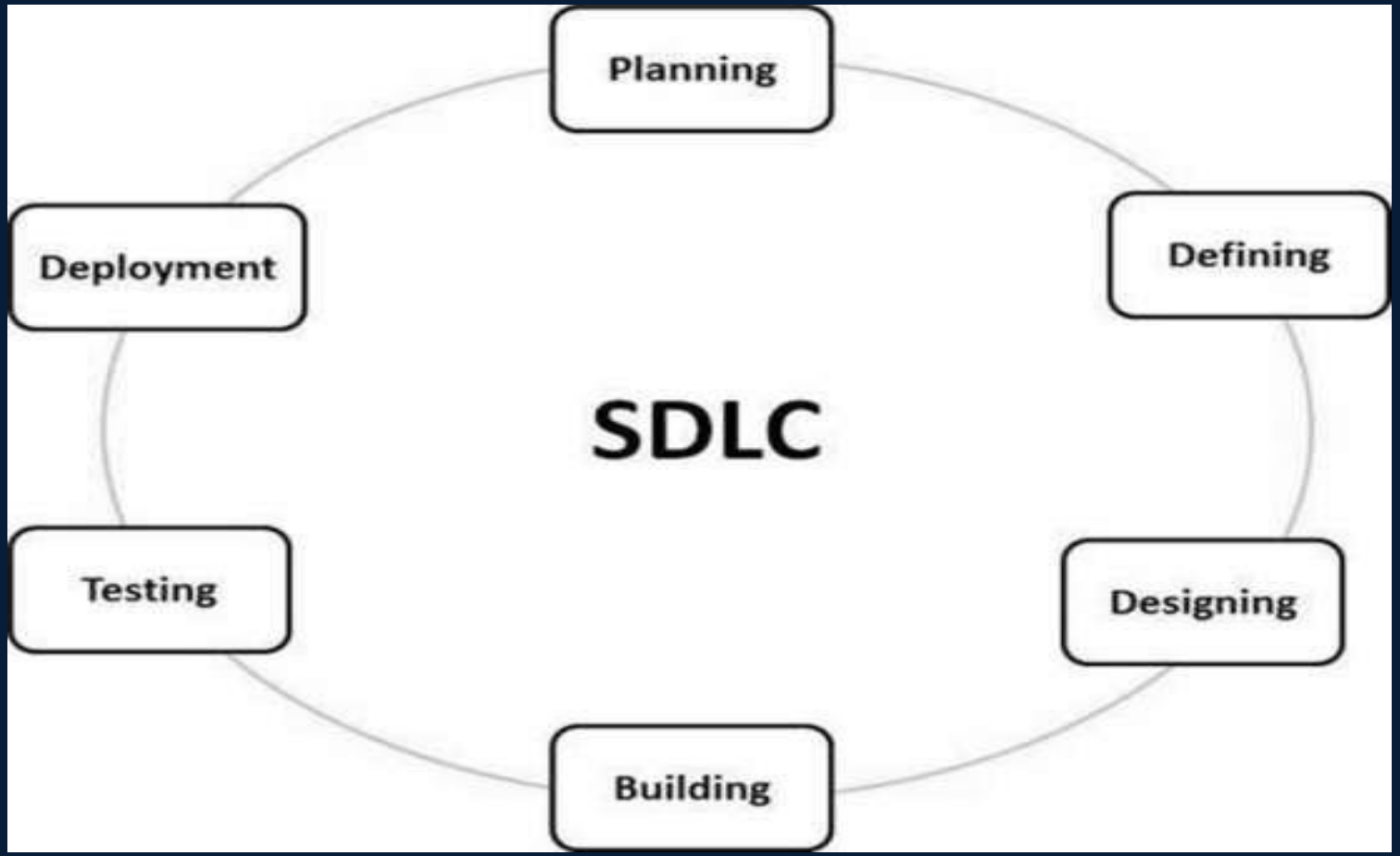
- *SDLC is a process followed for a software project, within a software organization. It consists of a detailed plan describing how to develop, maintain, replace and alter or enhance specific software. The life cycle defines a methodology for improving the quality of software and the overall development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

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.

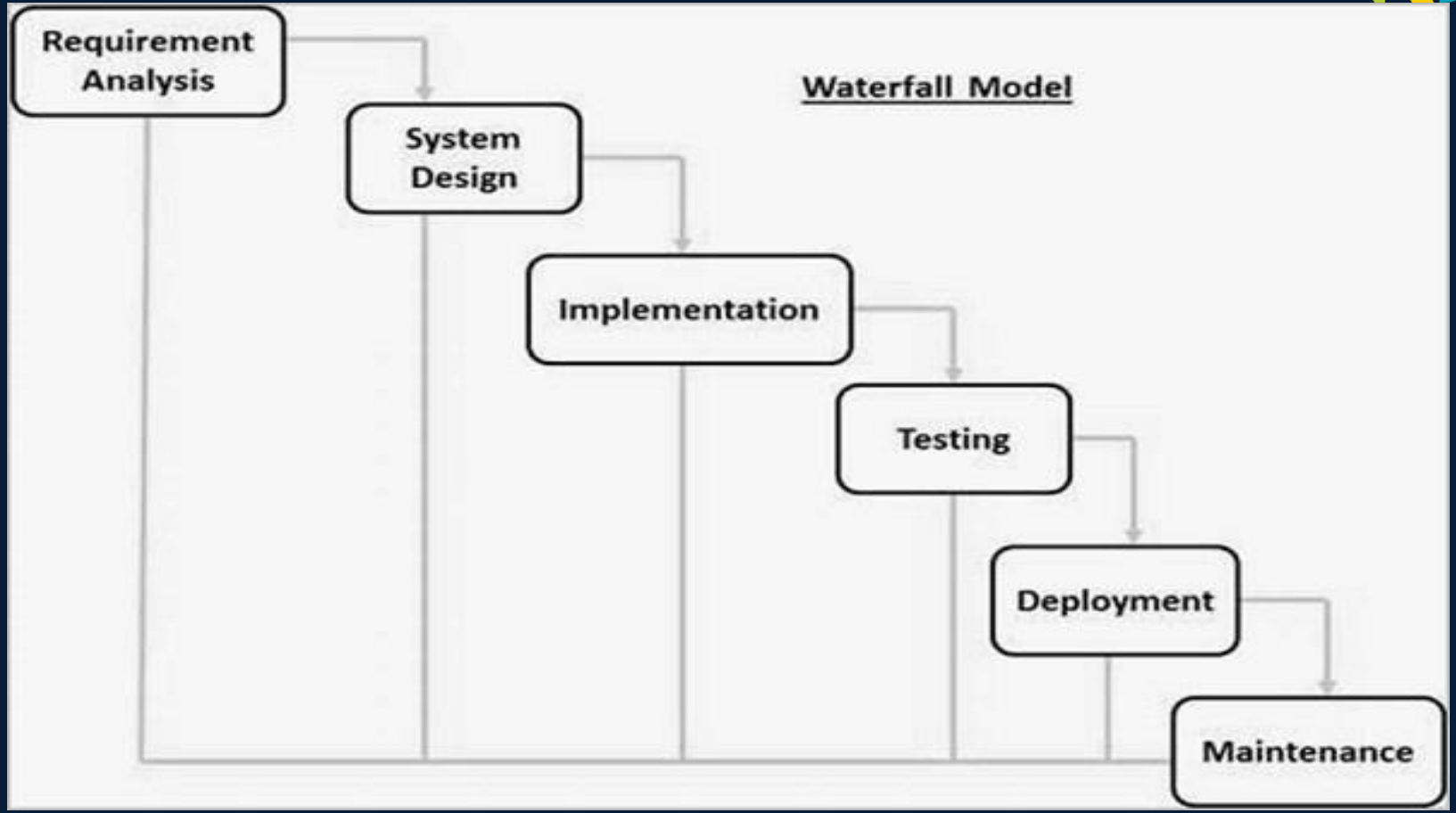
- *Following are the most important and popular SDLC models followed in the industry –*

❖ *Waterfall Model*
❖ *Prototype Model,*
❖ *Spiral Model,*
❖ *Evolutionary Development Models,*
❖ *Iterative Enhancement Models.*


Planning
Defining
Designing
Building
Testing
Deployment
SDLC

Waterfall Model

- *The Waterfall Model was the first Process Model to be introduced. It is also referred to as a linear-sequential life cycle model. It is very simple to understand and use. In a waterfall model, each phase must be completed before the next phase can begin and there is no overlapping in the phases.*
- *The Waterfall model is the earliest SDLC approach that was used for software development.*
- *In this Waterfall model, typically, the outcome of one phase acts as the input for the next phase sequentially.*
- *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
Requirement Analysis
System Design
Implementation
Testing
Deployment
Maintenance

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

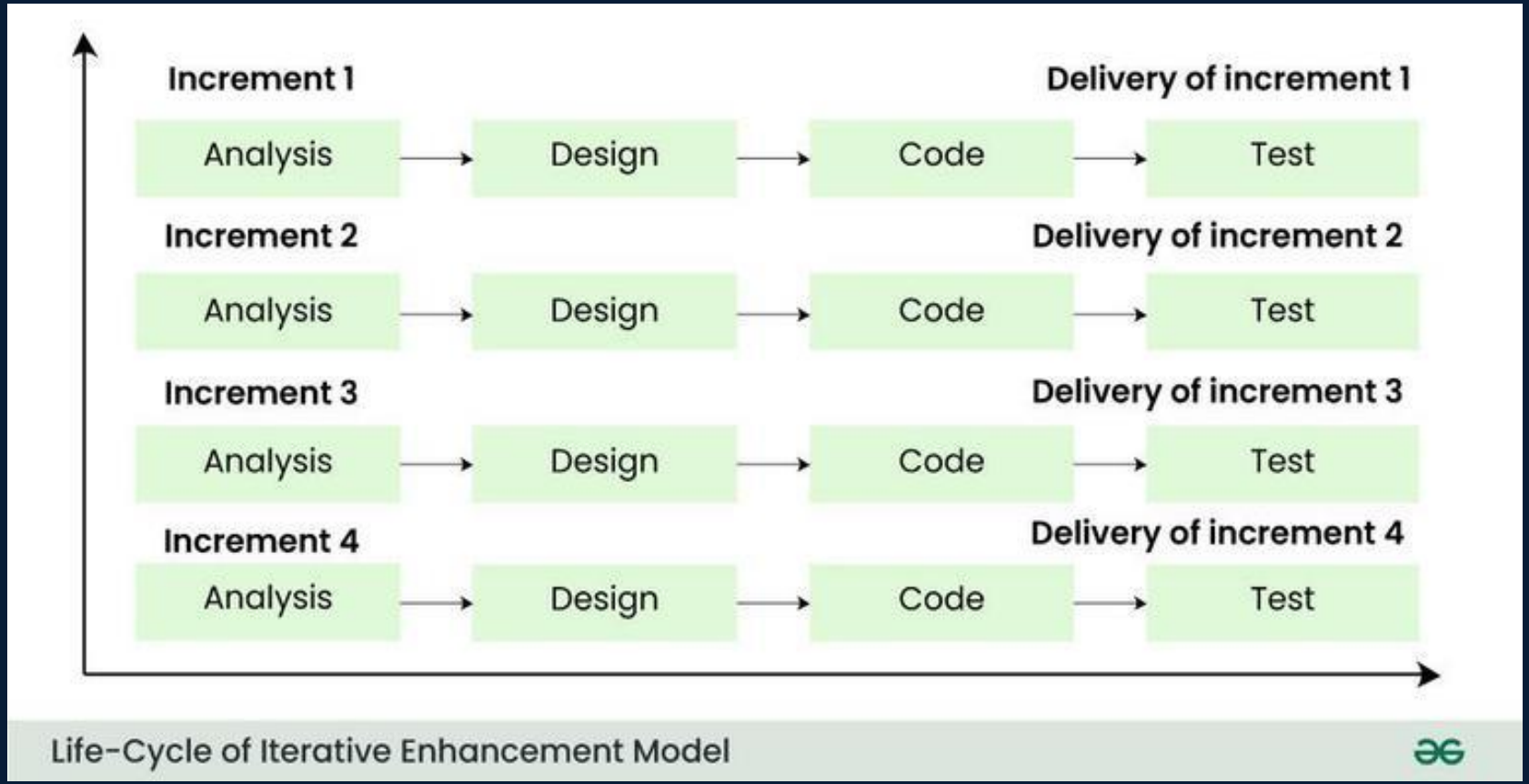
- Simple and easy to understand and use
- Easy to manage due to the rigidity of the model. Each phase has specific deliverables and a review process.
- Phases are processed and completed one at a time.
- Works well for smaller projects where requirements are very well understood.
- Clearly defined stages.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

Waterfall Model - Disadvantages

- 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 developement:*** *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:*** *Developement in e-commerece field often involves constant updates. Implementing an iterative approach enables the introduction of new functionality.*

Advantages of Iterative Enhancement Model

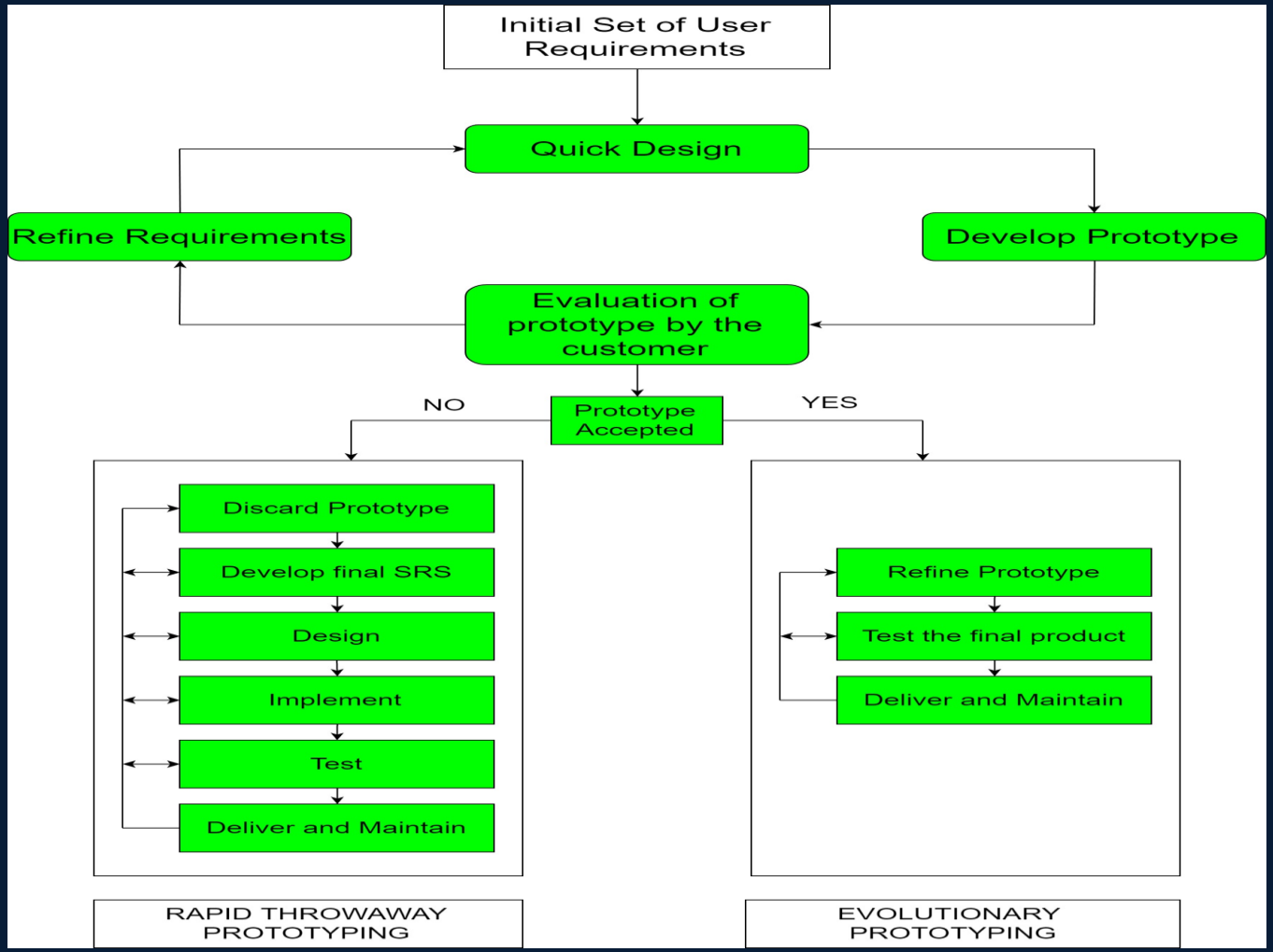
- *Adaptation to changing requirements is made possible by its flexibility in accommodating modifications and improvement during each iteration.*
- *Problems and risks can be identified and addressed early in the development process, reduces chances of issue for future stages.*
- *Every iteration is 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,*

Prototyping Model

- *Prototyping is defined as the process of developing a working replication of a product or system that has to be engineered. It offers a small-scale facsimile of the end product and is used for obtaining customer feedback.*
- *This model is used when the customers do not know the exact project requirements beforehand. In this model, a prototype of the end product is first developed, tested, and refined as per customer feedback repeatedly till a final acceptable prototype is achieved which forms the basis for developing the final product.*
- *In this process model, the system is partially implemented before or during the analysis phase thereby giving the customers an opportunity to see the product early in the life cycle. The process starts by interviewing the customers and developing the incomplete high-level paper model. Once the customer figures out the problems, the prototype is further refined to eliminate them.*


Initial Set of User Requirements
Quick Design
Refine Requirements
Develop Prototype
Evaluation of prototype by the customer
NO
Prototype Accepted
YES
Discard Prototype
Develop final SRS
Design
Implement
Test
Deliver and Maintain
Refine Prototype
Test the final product
Deliver and Maintain
RAPID THROWAWAY PROTOTYPING
EVOLUTIONARY PROTOTYPING

Applications of Prototyping Model

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

Advantages of Prototyping Model

- *The customers get to see the partial product early in the life cycle. This ensures a greater level of customer satisfaction and comfort.*
- *New requirements can be easily accommodated as there is scope for refinement.*
- *Missing functionalities can be easily figured out.*
- *Errors can be detected much earlier thereby saving a lot of effort and cost, besides enhancing the quality of the software.*

Disadvantages of the Prototyping Model

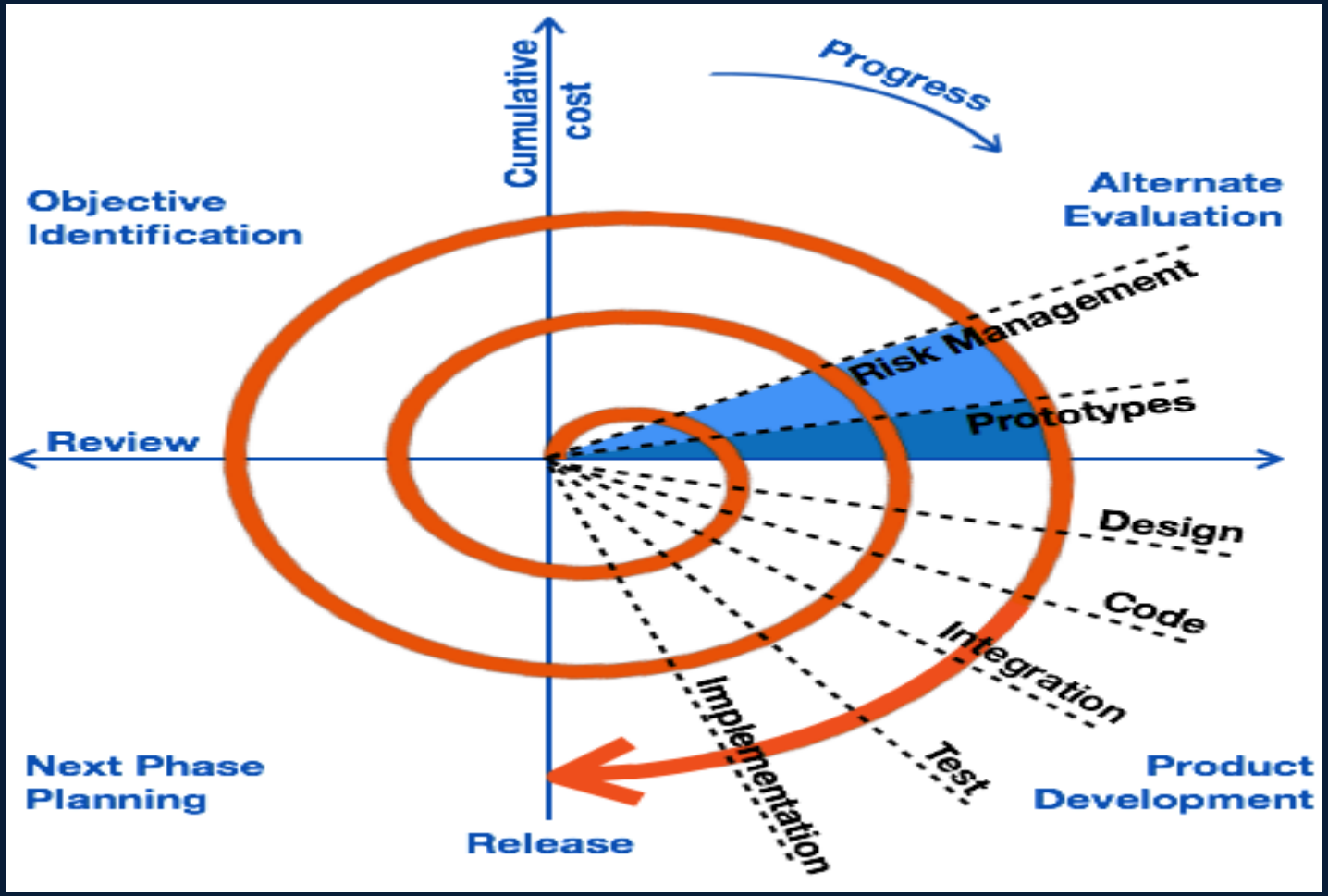
- *Costly with respect to time as well as money.*
- *There may be too much variation in requirements each time the prototype is evaluated by the customer.*
- *Poor Documentation due to continuously changing customer requirements.*
- *It is very difficult for developers to accommodate all the changes demanded by the customer.*

SDLC - Spiral Model

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

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

Identification, Design, Construct or Build and Evaluation and Risk Analysis


Cumulative
cost
Progress
Objective
Identification
Alternate
Evaluation
Risk Management
Prototypes
Review
Design
Code
Integration
Implementation
Test
Next Phase
Planning
Product
Development
Release

Where we USE

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

Spiral Model - Pros and Cons

The advantages of the Spiral SDLC Model are as follows –

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

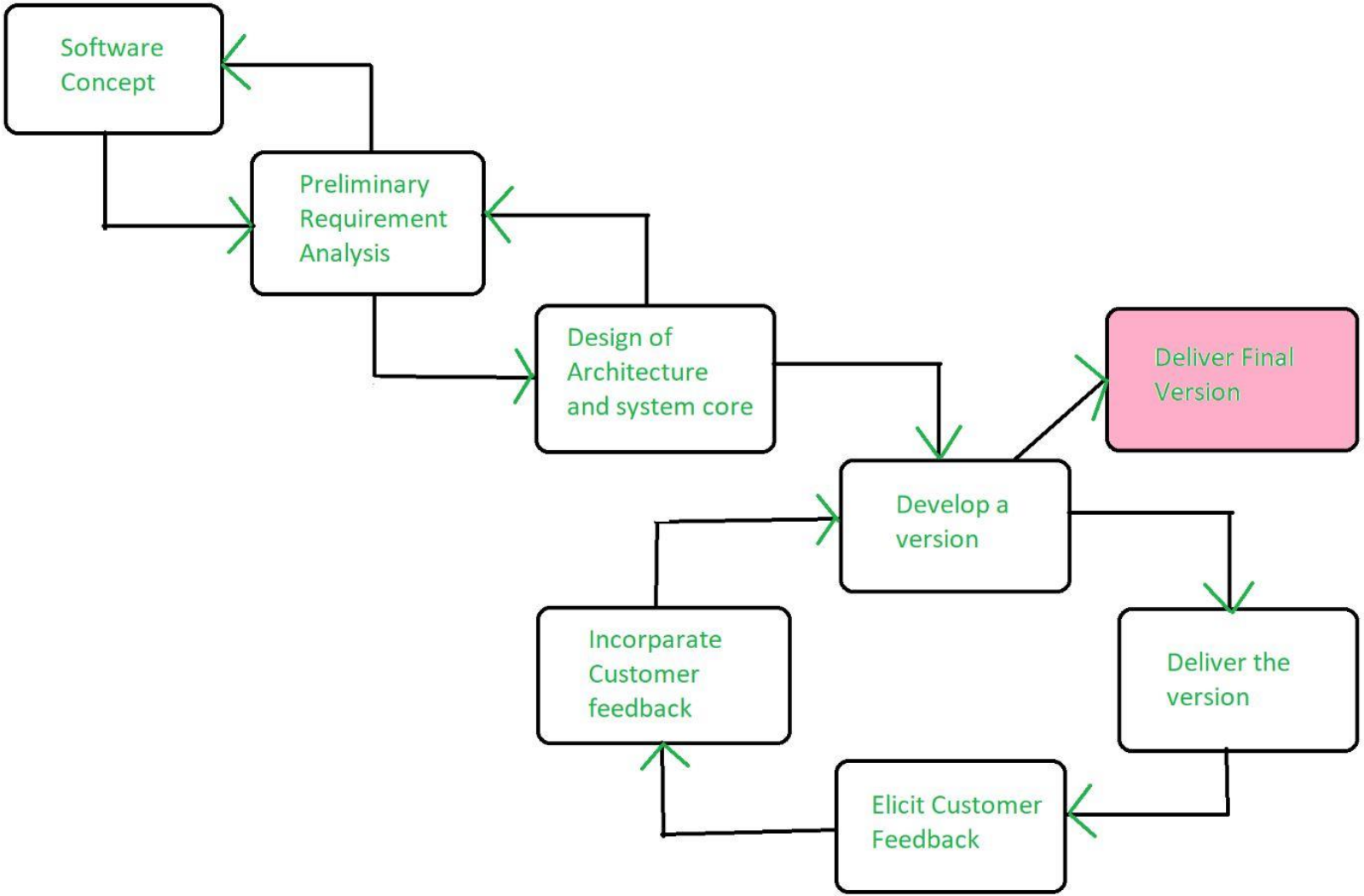
The disadvantages of the Spiral SDLC Model are as follows –

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

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.


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

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.*

Thank You


SR
UNIVERSITY

Software Testing Methodologies

Term: 2023-24

Unit-2

Text Books: 1. Software Engineering, A practitioner's approach
Roger s. Pressman 6th edition McGraw-Hill
2. Software Engineering Somerville 7th edition

Unit-2 Syllabus

Design Engineering: Design process and design quality, design concepts, the design model. **Creating an architectural design:** software architecture, data design, architectural styles and patterns, architectural design, **conceptual model of UML**, basic structural modeling, class diagrams, sequence diagrams, collaboration diagrams, use case diagrams, component diagrams.

Introduction

Software Design is the process of transforming user requirements into a suitable form, which helps the programmer in software coding and testing. During the software design phase, the design document is produced, based on the customer requirements as documented in the SRS document. Hence, this phase aims to transform the SRS document into a design document.

It is iterative process through which requirements are translated into a " blue print" for constructing the software.

The following items are designed and documented during the design phase:

- Different modules are required.
- Control relationships among modules.
- Interface among different modules.
- Data structure among the different modules.
- Algorithms are required to be implemented among the individual modules.

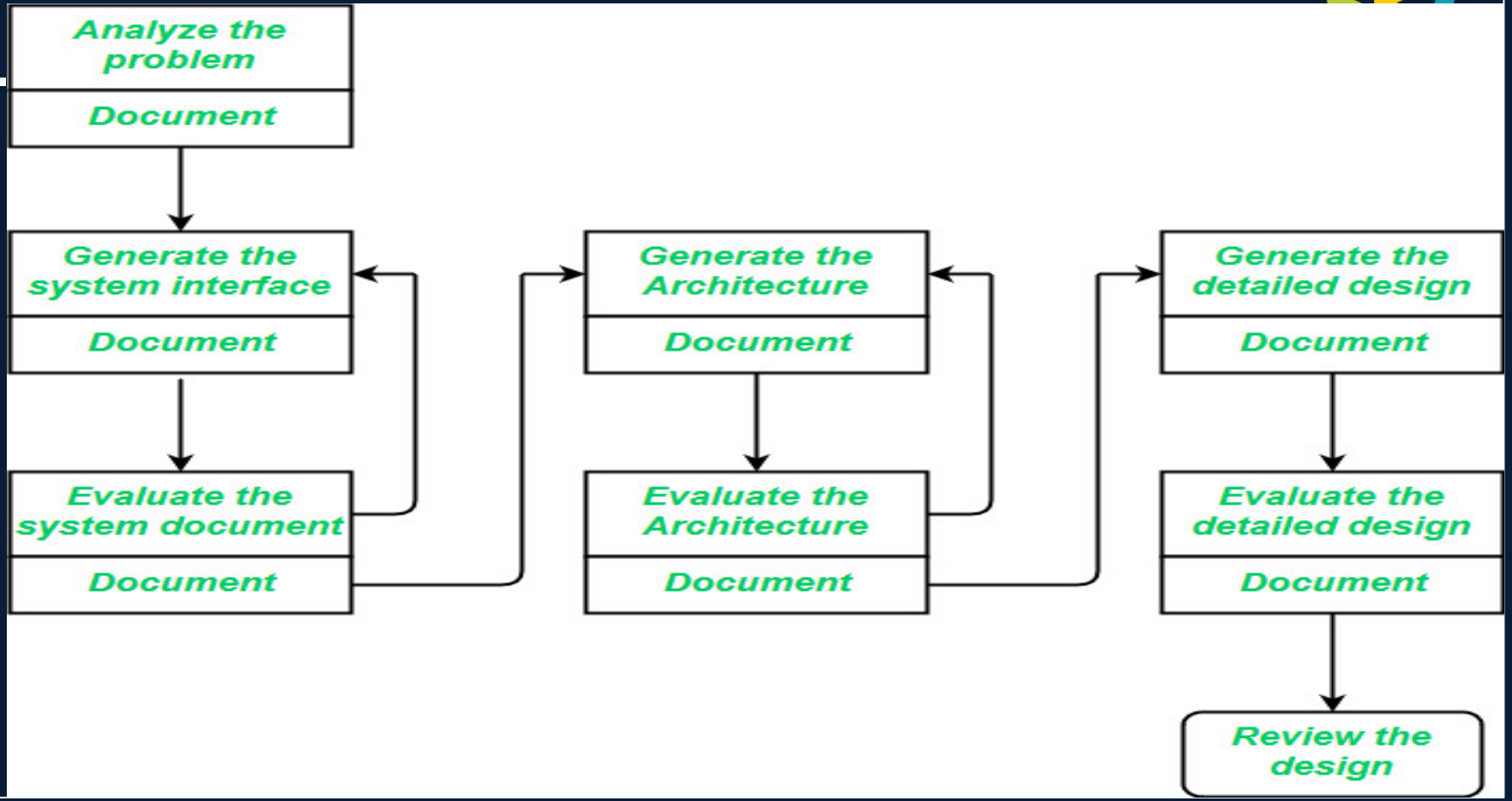
The software design process can be divided into the following three levels or phases of design:

- Interface Design
- Architectural Design
- Detailed Design

Objectives of Software Design

Objectives of Software Design

1. Correctness: Software design should be correct as per requirement.
2. Completeness: The design should have all components like data structures, modules, and external interfaces, etc.
3. Efficiency: Resources should be used efficiently by the program.
4. Flexibility: Able to modify on changing needs.
5. Consistency: There should not be any inconsistency in the design.
6. Maintainability: The design should be so simple so that it can be easily maintainable by other designers.


Analyze the problem
Document
Generate the system interface
Document
Evaluate the system document
Document
Generate the Architecture
Document
Evaluate the Architecture
Document
Generate the detailed design
Document
Evaluate the detailed design
Document
Review the design

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 concerning the inner workings of the system. The design problem statement produced during the problem analysis step should identify the people, other systems, and devices that are collectively called agents.

Interface design should include the following details:

- A Precise description of events in the environment, or messages from agents to which the system must respond.
- A Precise description of the events or messages that the system must produce.
- Specification of the data, and the formats of the data coming into and going out of the system.
- 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:

- Gross decomposition of the systems into major components.
- Allocation of functional responsibilities to components.
- Component Interfaces.
- Component scaling and performance properties, resource consumption properties, reliability properties, and so forth.
- 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 data structures.

The detailed design may include:

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

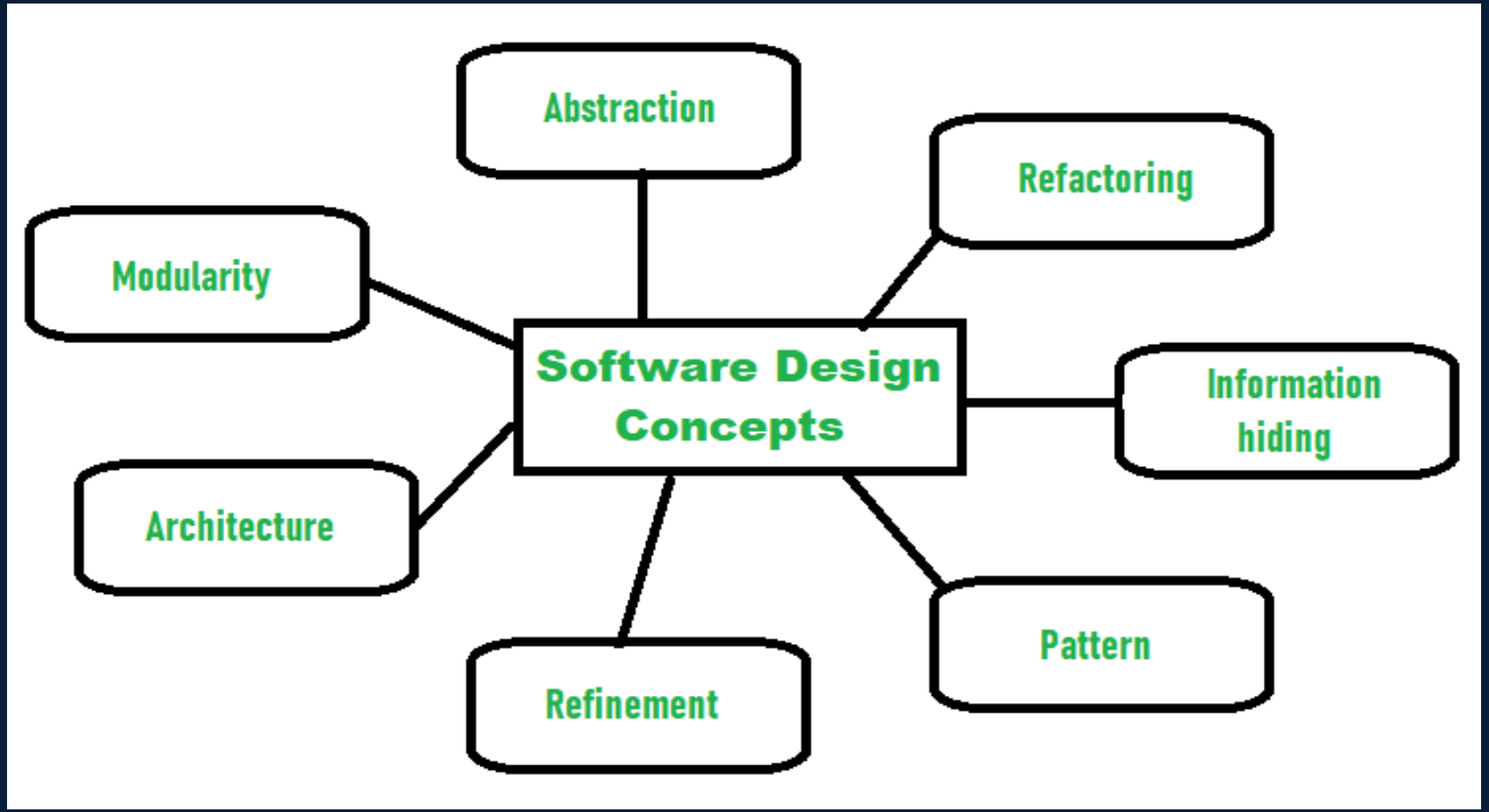
Design Concepts

- Concepts are defined as a principal idea or invention that comes into our mind or in thought to understand something. The software design concept simply means the idea or principle behind the design.

- It describes how you plan to solve the problem of designing software, the logic, or thinking behind how you will design software. It allows the software engineer to create the model of the system or software or product that is to be developed or built.

- The software design concept provides a supporting and essential structure or model for developing the right software. There are many concepts of software design and some of them are given below:

Design Concepts



Design Model

- The design model builds on the analysis model by describing, in greater detail, the structure of the system and how the system will be implemented. Classes that were identified in the analysis model are refined to include the implementation constructs.
- In the design model, packages contain the design elements of the system, such as design classes, interfaces, and design subsystems, that evolve from the analysis classes. Each package can contain any number of sub packages that further partition the contained design elements. These architectural layers form the basis for a second-level organization of the elements that describe the specifications and implementation details of the system.

- ✓ Within each package, sequence diagrams illustrate how the objects in the classes interact, state machine diagrams to model the dynamic behavior in classes, component diagrams to describe the software architecture of the system, and deployment diagrams to describe the physical architecture of the system.

- ✓ The design model is based on the analysis and architectural requirements of the system. It represents the application components and determines their appropriate placement and use within the overall architecture.

Software Architecture

- The complete structure of the software is known as software architecture.

- Structure provides conceptual integrity for a system in a number of ways.

- The architecture is the structure of program modules where they interact with each other in a specialized way.
- The components use the structure of data.
- The aim of the software design is to obtain an architectural framework of a system.
- The more detailed design activities are conducted from the framework.

Data Design

- **Data design** is the first design activity, which results in less complex, modular and efficient program structure. The information domain model developed during analysis phase is transformed into data structures needed for implementing the software. The data objects, attributes, and relationships depicted in entity relationship diagrams and the information stored in data dictionary provide a base for data design activity. During the data design process, data types are specified along with the integrity rules required for the data.

The structure of data can be viewed at three levels, *namely, program component level, application level, and business level*. At the **program component level**, the design of data structures and the algorithms required to manipulate them is necessary, if high-quality software is desired. At the **application level**, it is crucial to convert the data model into a database so that the specific business objectives of a system could be achieved. At the **business level**, the collection of information stored in different databases should be reorganized into data warehouse, which enables data mining that has an influential impact on the business.

Architectural Patterns and Styles

An architectural Style is a specialization of element and relation types, together with a set of constraints on how they can be used.

On the other hand, an architectural Pattern expresses a fundamental structural organization schema for software systems. It provides a set of predefined subsystems, specifies their responsibilities, and includes rules and guidelines for organizing the relationships between them.

It argues that a Pattern is a context-problem-solution triple; a Style is simply a condensation that focuses most heavily on the solution part.

Architectural styles and patterns are the same things. But how can it be? The word style means: **“a manner of doing something”** while a pattern means: **“a repeated decorative design”**.

The architectural style shows how do we organize our code, or how the system will look like from 10000 feet helicopter view to show the highest level of abstraction of our system design. Furthermore, when building the architectural style of our system we focus on layers and modules and how they are communicating with each other. There are different types of architectural styles, and moreover, we can mix them and produce a hybrid style that consists of a mix between two and even more architectural styles.

- The architectural style shows how do we organize our code, or how the system. Below is a list of architectural styles and examples for each category:
- **Structure architectural styles:** such as layered, pipes and filters and component-based styles.
- **Messaging styles:** such as Implicit invocation, asynchronous messaging and publish-subscribe style.
- **Distributed systems:** such as service-oriented, peer to peer style, object request broker, and cloud computing styles.
- **Shared memory styles:** such as role-based, blackboard, database-centric styles.
- **Adaptive system styles:** such as microkernel style, reflection, domain-specific language styles.

Architectural Patterns

The architectural pattern shows how a solution can be used to solve a reoccurring problem. In another word, it reflects how a code or components interact with each other.

Moreover, the architectural pattern is describing the architectural style of our system and provides solutions for the issues in our architectural style. Personally, I prefer to define architectural patterns as a way to implement our architectural style.

For example:

- ✓ how to separate the UI of the data module in our architectural style? How to integrate a third-party component with our system? how many tires will we have in our client-server architecture?
- ✓ Examples of architectural patterns are microservices, message bus, service requester/ consumer, MVC, MVVM, microkernel, n-tier, domain-driven design, and presentation-abstraction-control.

Architectural design

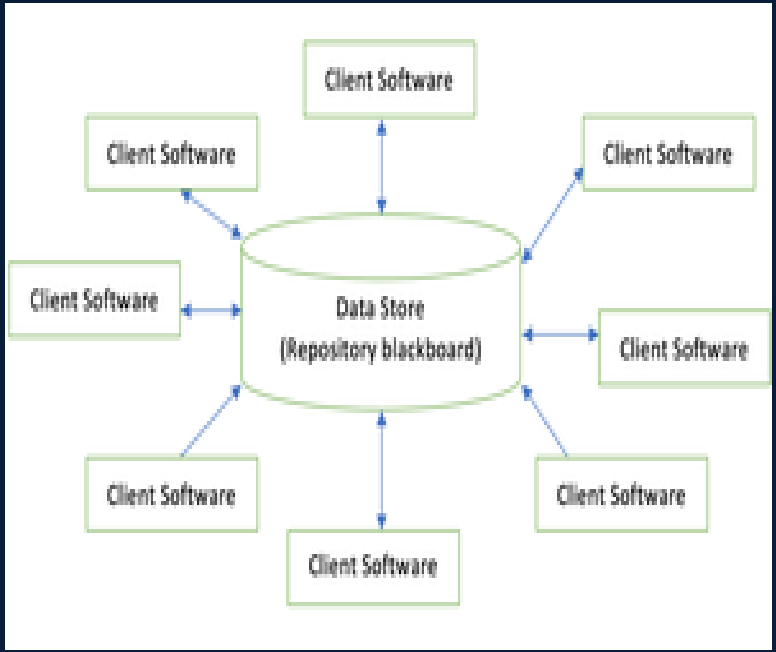
The software needs the architectural design to represents the design of software. IEEE defines architectural design as “the process of defining a collection of hardware and software components and their interfaces to establish the framework for the development of a computer system.” The software that is built for computer-based systems can exhibit one of these many architectural styles.

Each style will describe a system category that consists of :

- A set of components(eg: a database, computational modules) that will perform a function required by the system.
- The set of connectors will help in coordination, communication, and cooperation between the components.
- Conditions that how components can be integrated to form the system.
- Semantic models that help the designer to understand the overall properties of the system.

Data centered architectures

- A data store will reside at the center of this architecture and is accessed frequently by the other components that update, add, delete or modify the data present within the store.
- The figure illustrates a typical data centered style. The client software access a central repository. Variation of this approach are used to transform the repository into a blackboard when data related to client or data of interest for the client change the notifications to client software.
- This data-centered architecture will promote integrability. This means that the existing components can be changed and new client components can be added to the architecture without the permission or concern of other clients.
- Data can be passed among clients using blackboard mechanism.

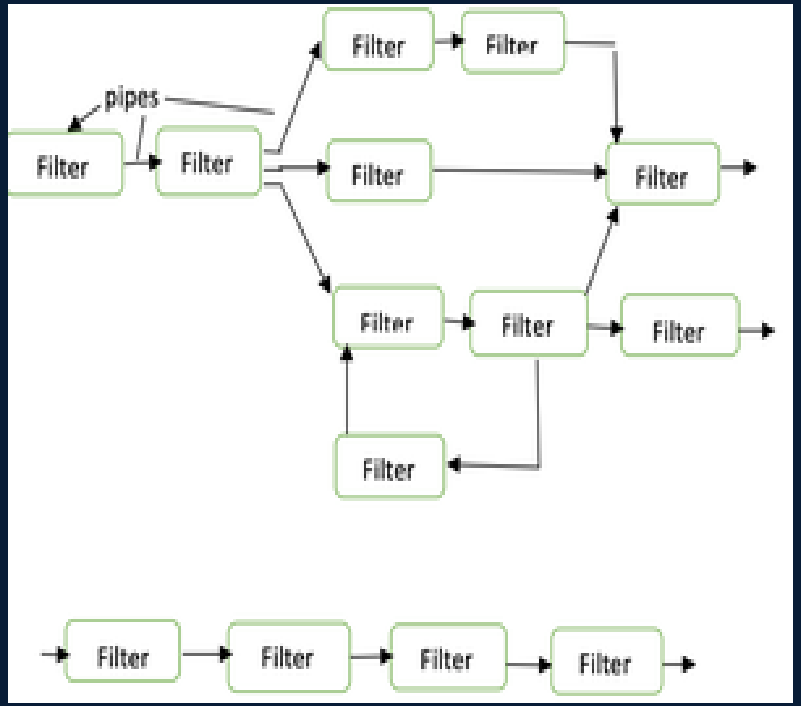


Advantages

- ✓ Repository of data is independent of clients
- ✓ Client work independent of each other
- ✓ It may be simple to add additional clients.
- ✓ Modification can be very easy

Data flow architectures

- This kind of architecture is used when input data is transformed into output data through a series of computational manipulative components.
- The figure represents pipe-and-filter architecture since it uses both pipe and filter and it has a set of components called filters connected by lines.
- Pipes are used to transmitting data from one component to the next.
- Each filter will work independently and is designed to take data input of a certain form and produces data output to the next filter of a specified form. The filters don’t require any knowledge of the working of neighboring filters.
- If the data flow degenerates into a single line of transforms, then it is termed as batch sequential. This structure accepts the batch of data and then applies a series of sequential components to transform it.



Advantages of Data Flow architecture

- It encourages upkeep, repurposing, and modification.
- With this design, concurrent execution is supported.

The disadvantage of Data Flow architecture

- It frequently degenerates to batch sequential system
- Data flow architecture does not allow applications that require greater user engagement.
- It is not easy to coordinate two different but related streams

Object Oriented architecture:

The components of a system encapsulate data and the operations that must be applied to manipulate the data. The coordination and communication between the components are established via the message passing.

Characteristics of Object Oriented architecture

- Object protect the system's integrity.
- An object is unaware of the depiction of other items.

Advantage of Object Oriented architecture

- It enables the designer to separate a challenge into a collection of autonomous objects.
- Other objects are aware of the implementation details of the object, allowing changes to be made without having an impact on other objects.

Layered architecture

- A number of different layers are defined with each layer performing a well-defined set of operations. Each layer will do some operations that becomes closer to machine instruction set progressively.
- At the outer layer, components will receive the user interface operations and at the inner layers, components will perform the operating system interfacing(communication and coordination with OS)
- Intermediate layers to utility services and application software functions.
- One common example of this architectural style is OSI-ISO (Open Systems Interconnection-International Organisation for Standardisation) communication system.


CLIENT
APPLICATION
PLATFORM
INFRASTRUCTURE
SERVER

Thank You

UNIT-II

Design Engineering:

Design process –

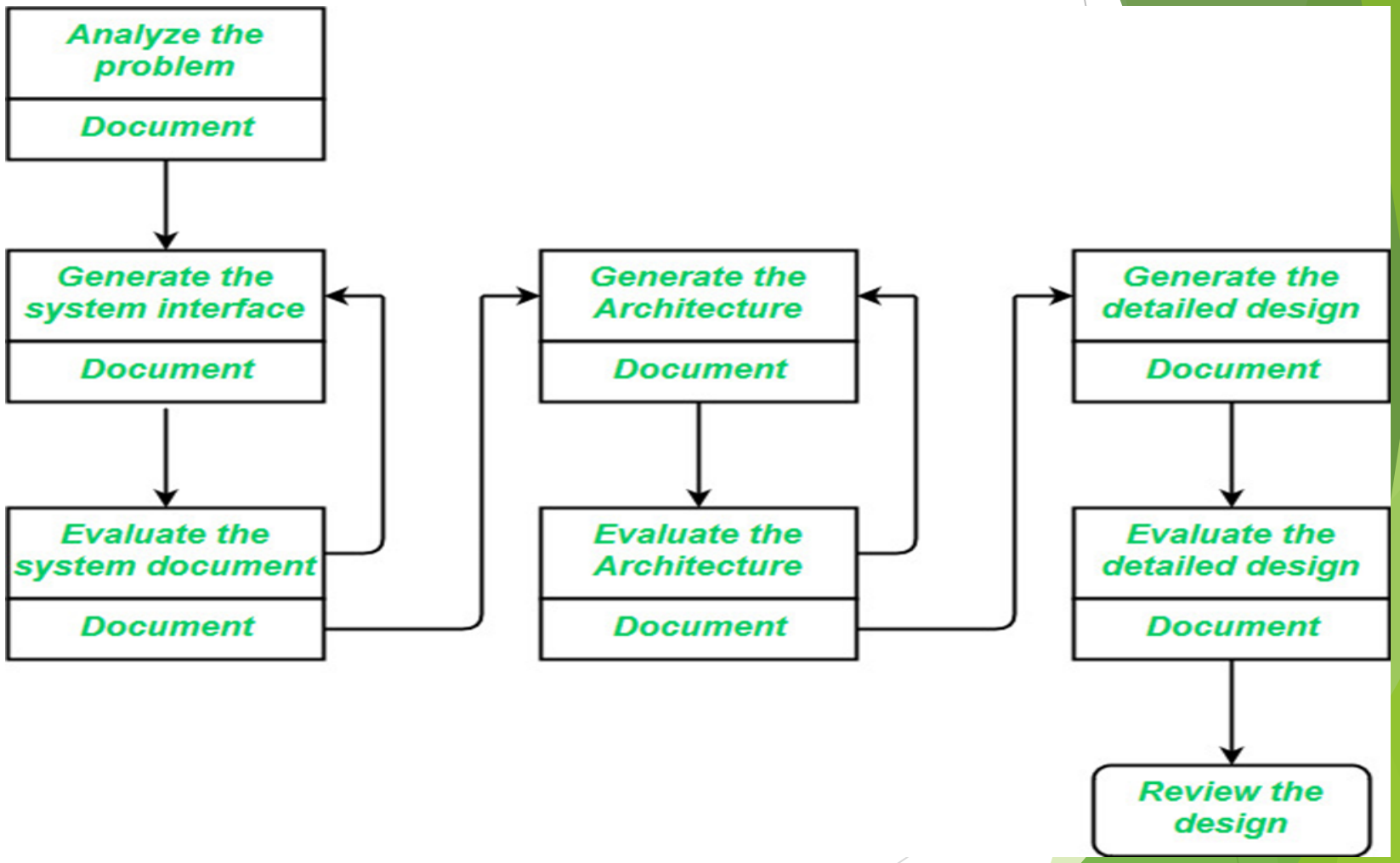
The design phase of software development deals with transforming the customer requirements as described in the SRS (Software Requirement Specifications) documents into a form implementable using a programming language.

Elements of a System

2. **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.
3. **Modules:** These are components that handle one specific task in a system. A combination of the modules makes up the system.
4. **Components:** This provides a particular function or group of related functions. They are made up of modules.
5. **Interfaces:** This is the shared boundary across which the components of a system exchange information and relate.
6. **Data:** This is the management of the information and data flow.

The software design process can be divided into the following three levels or phases of design:

2. Interface Design
3. Architectural Design
4. Detailed Design Process



Interface Design

Interface design specifies the interaction between a system and its environment. This phase proceeds at a high level of abstraction concerning the inner workings of the system i.e, during interface design, the internals 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:

2. Precise(accurate) description of events in the environment, or messages from agents to which the system must respond.
3. Precise description of the events or messages that the system must produce.
4. Specification of the data, and the data formats coming into and going out of the system.
5. 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:

2. Gross decomposition of the systems into major components.
3. Allocation of functional responsibilities to components.
4. Component Interfaces.
5. Component scaling and performance properties, resource consumption properties, reliability properties, and so forth.
6. 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:

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

Software design quality -

Objectives of Software Design

Objectives of software quality design

1.Correctness: Software design should be correct as per requirement.

2.Completeness: The design should have all components like data structures, modules, external interfaces, etc.

3.Efficiency: Resources should be used efficiently by the program.

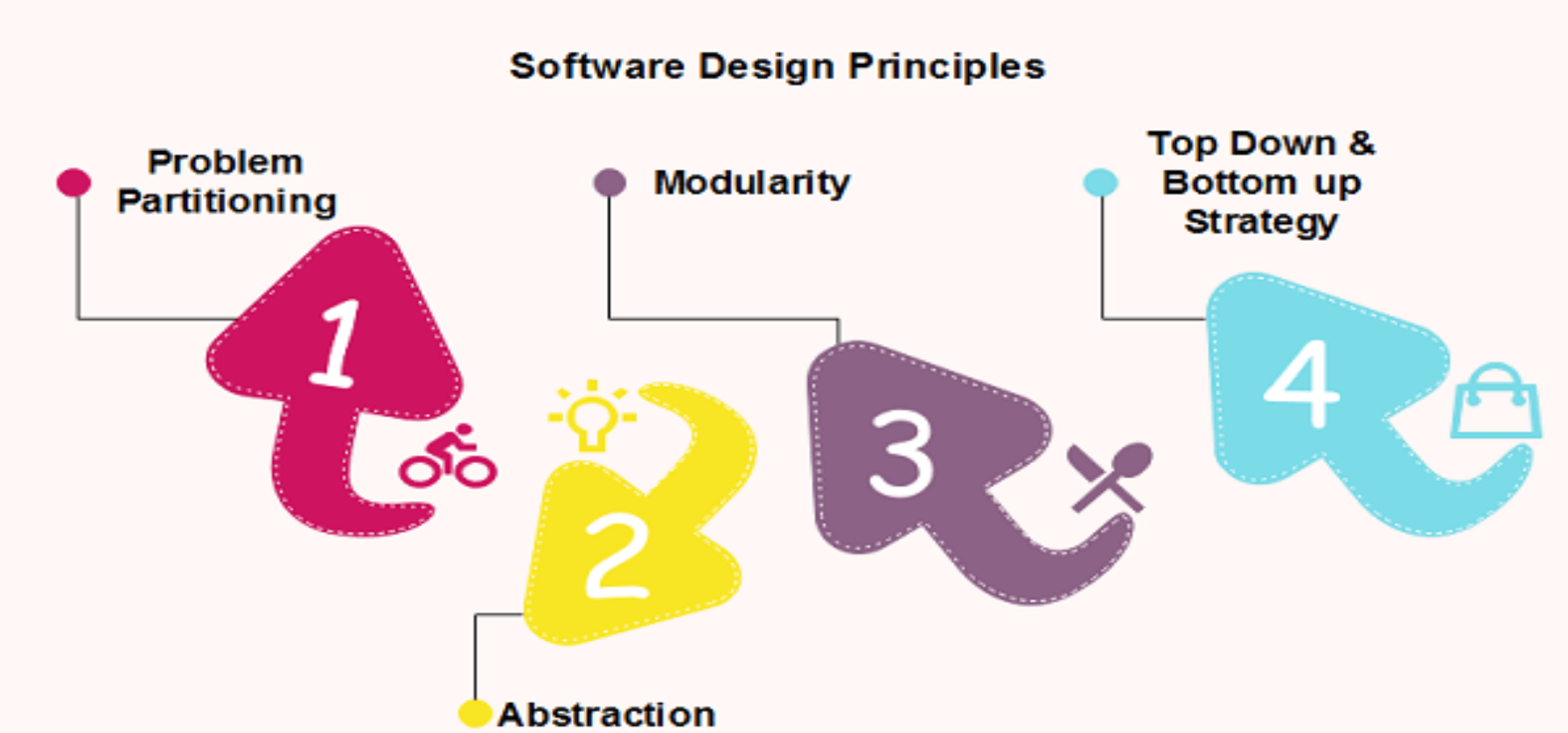
4.Flexibility: Able to modify on changing needs.

5.Consistency: There should not be any inconsistency in the design.

6.Maintainability: The design should be so simple so that it can be easily maintainable by other designers.

Software Design Concepts (principles)

Every software process is characterized by basic concepts along with certain practices or methods.



Problem Partitioning

For small problem, we can handle the entire problem at once but for the significant problem, divide the problems and conquer the problem it means to divide the problem into smaller pieces so that each piece can be captured separately.

For software design, the goal is to divide the problem into manageable pieces.

Benefits of Problem Partitioning

1.Software is easy to understand
2.Software becomes simple
3.Software is easy to test
4.Software is easy to modify
5.Software is easy to maintain
6.Software is easy to expand

Abstraction

An abstraction is a tool that enables a designer to consider a component at an abstract level without bothering about the internal details of the implementation. Abstraction can be used for existing element as well as the component being designed.

Here, there are two common abstraction mechanisms

1.Functional Abstraction

2.Data Abstraction

Functional Abstraction

i.A module is specified by the method it performs.

ii.The details of the algorithm to accomplish the functions are not visible to the user of the function.

Functional abstraction forms the basis for **Function oriented design approaches**.

Data Abstraction

Details of the data elements are not visible to the users of data. Data Abstraction forms the basis for **Object Oriented design approaches**.

Modularity

Modularity specifies to the division of software into separate modules which are differently named and addressed and are integrated later on in to obtain the completely functional software. It is the only property that allows a program to be intellectually manageable. Single large programs are difficult to understand and read due to a large number of reference variables, control paths, global variables, etc.

Advantages of Modularity

There are several advantages of Modularity

- It allows large programs to be written by several or different people
- It encourages the creation of commonly used routines to be placed in the library and used by other programs.
- It simplifies the overlay procedure of loading a large program into main storage.
- It provides more checkpoints to measure progress.
- It provides a framework for complete testing, more accessible to test
- It produced the well designed and more readable program.

Disadvantages of Modularity

There are several disadvantages of Modularity

- Execution time maybe, but not certainly, longer
- Storage size perhaps, but is not certainly, increased
- Compilation and loading time may be longer
- Inter-module communication problems may be increased
- More linkage required, run-time may be longer, more source lines must be written, and more documentation has to be done

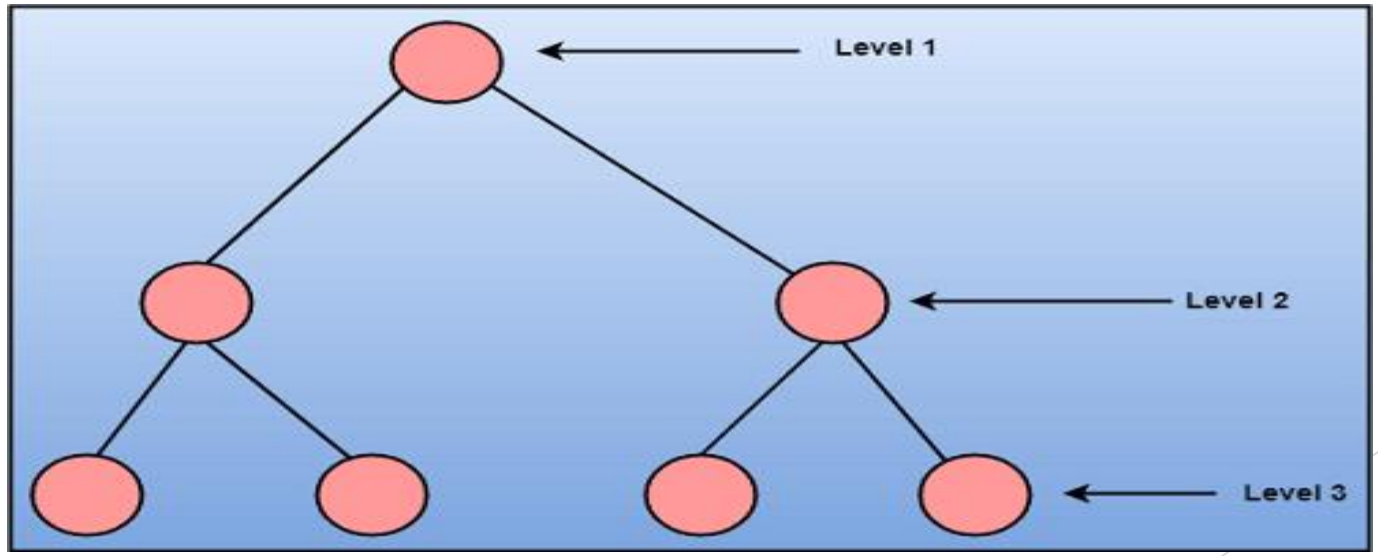
Strategy of Design

A good system design strategy is to organize the program modules in such a method that are easy to develop and latter too, change. Structured design methods help developers to deal with the size and complexity of programs. Analysts generate instructions for the developers about how code should be composed and how pieces of code should fit together to form a program.

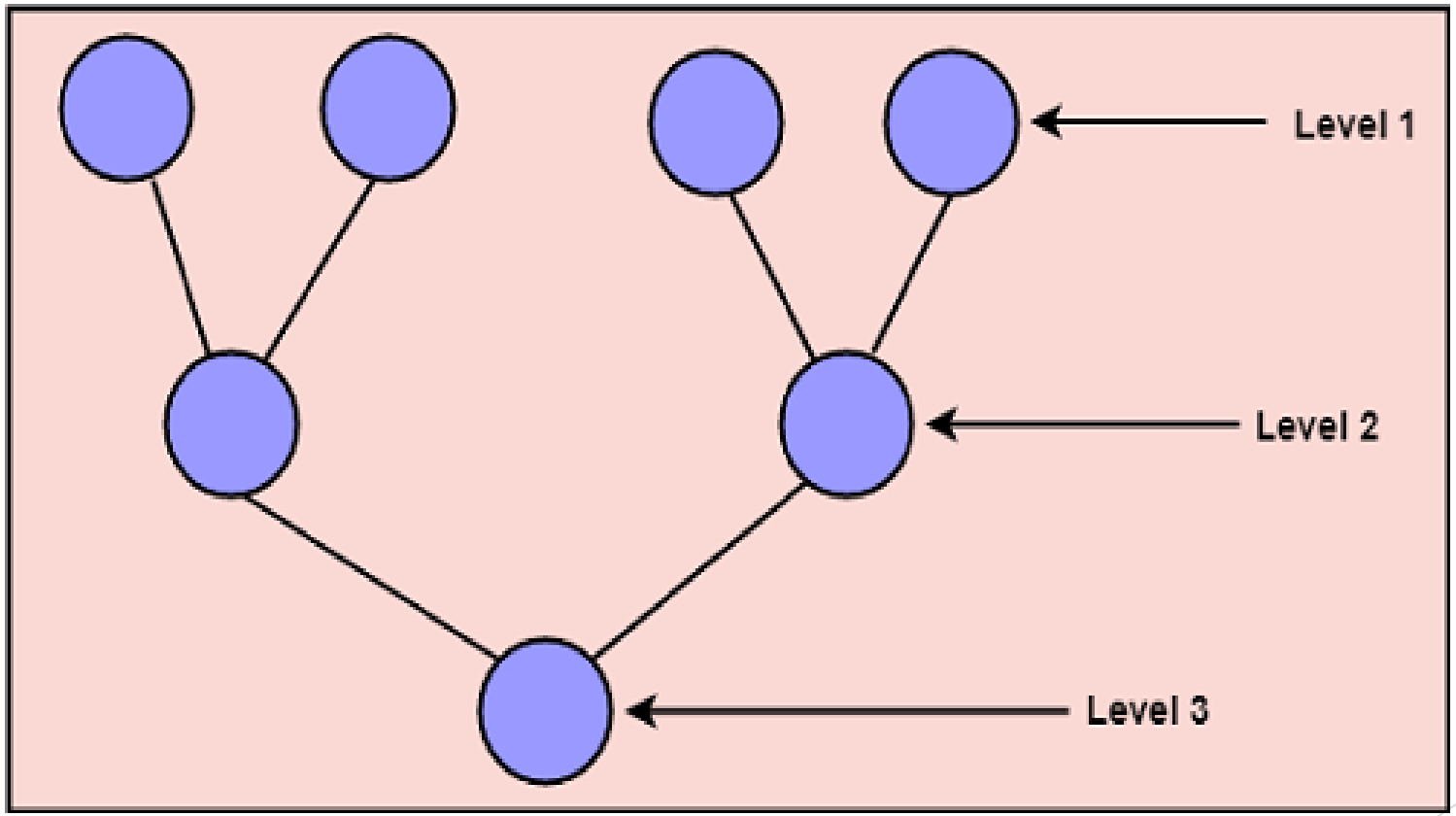
To design a system, there are two possible approaches:

1.Top-down Approach

This approach starts with the identification of the main components and then decomposing them into their more detailed sub-components.



2. Bottom-up Approach: A bottom-up approach begins with the lower details and moves towards up the hierarchy, as shown in fig. This approach is suitable in case of an existing system.



Types of Design Patterns


SR
UNIVERSITY

Software Testing Methodologies

Term: 2023-24

Unit-3

Text Books: 1. Software Testing Techniques - Boris Beizer, Dreamtech, 5th edition (2020).

Unit-3 Syllabus

Introduction: Testing definition, Purpose of Testing, Dichotomies, Testing Objectives, Faults, Errors, Bugs, and Failures, Software Testing Life Cycle (STLC), Verification, Validation and Testing, Types of testing, Software Quality and Reliability, Software defect, Consequences of Bugs, Taxonomy of Bugs.

Introduction

- ✓ Software testing can be stated as the process of verifying and validating whether a software or application is bug-free, meets the technical requirements as guided by its design and development, and meets the user requirements effectively and efficiently by handling all the exceptional and boundary cases.

- ✓ The process of software testing aims not only at finding faults in the existing software but also at finding measures to improve the software in terms of efficiency, accuracy, and usability.

- ✓ Software Testing is a method to assess the functionality of the software program. The process checks whether the actual software matches the expected requirements and ensures the software is bug-free.

Purpose of the Testing

The purpose of software testing is to identify the errors, faults, or missing requirements in contrast to actual requirements. It mainly aims at measuring the specification, functionality, and performance of a software program or application.

Software testing can be divided into two steps:

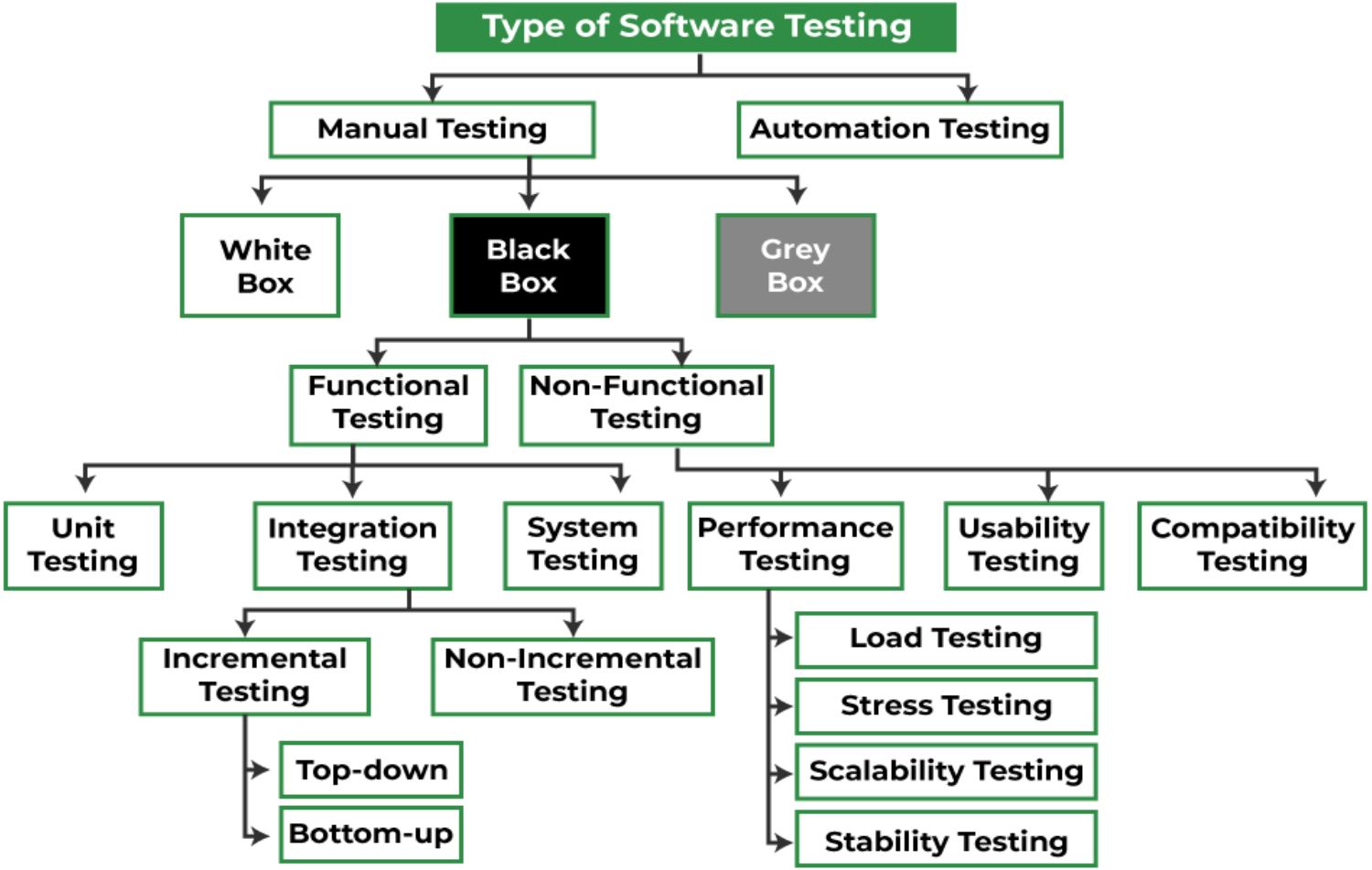
- **Verification:** It refers to the set of tasks that ensure that the software correctly implements a specific function. It means "Are we building the product right?".
- **Validation:** It refers to a different set of tasks that ensure that the software that has been built is traceable to customer requirements. It means "Are we building the right product?".

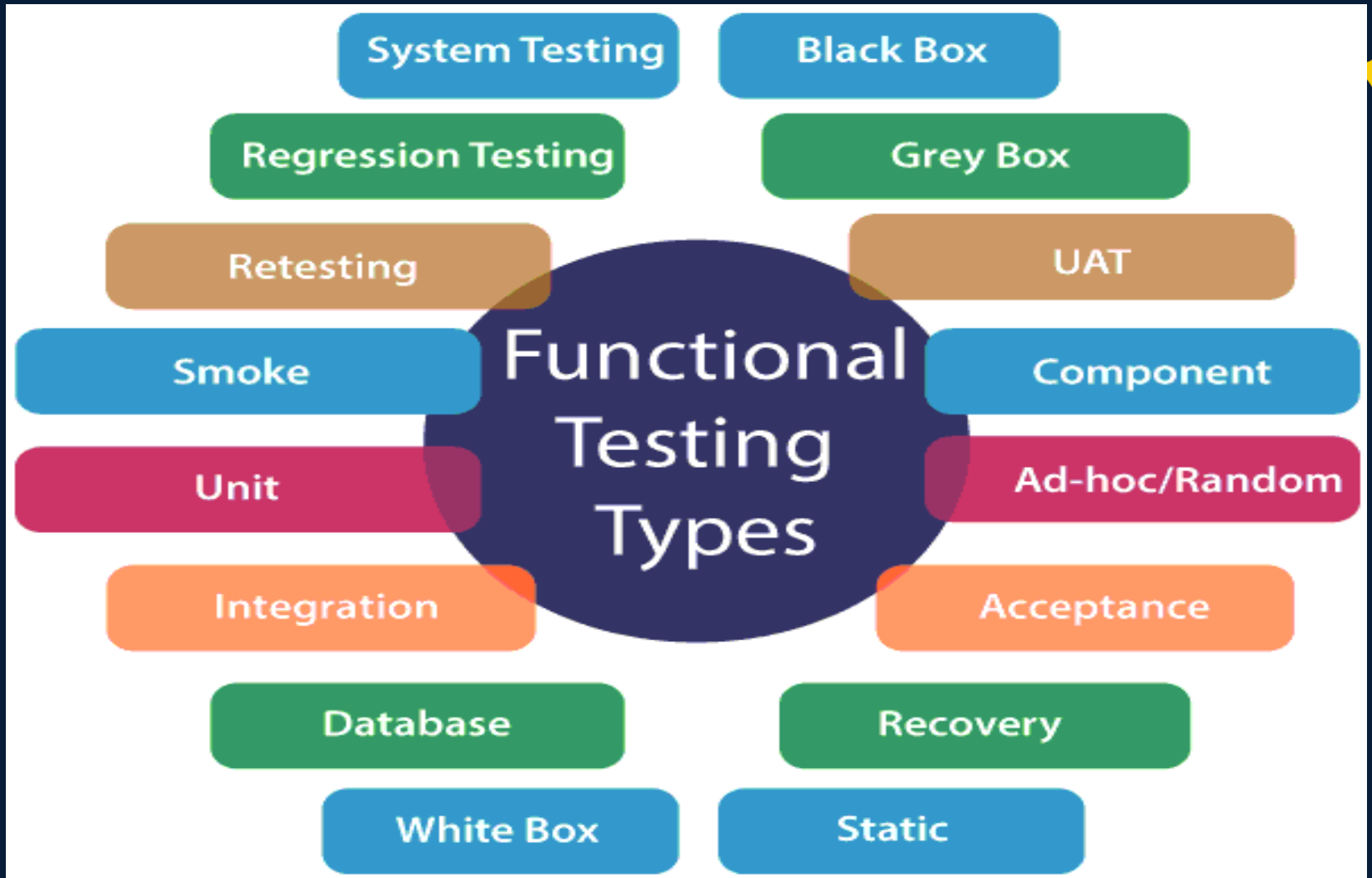
Purpose of Testing

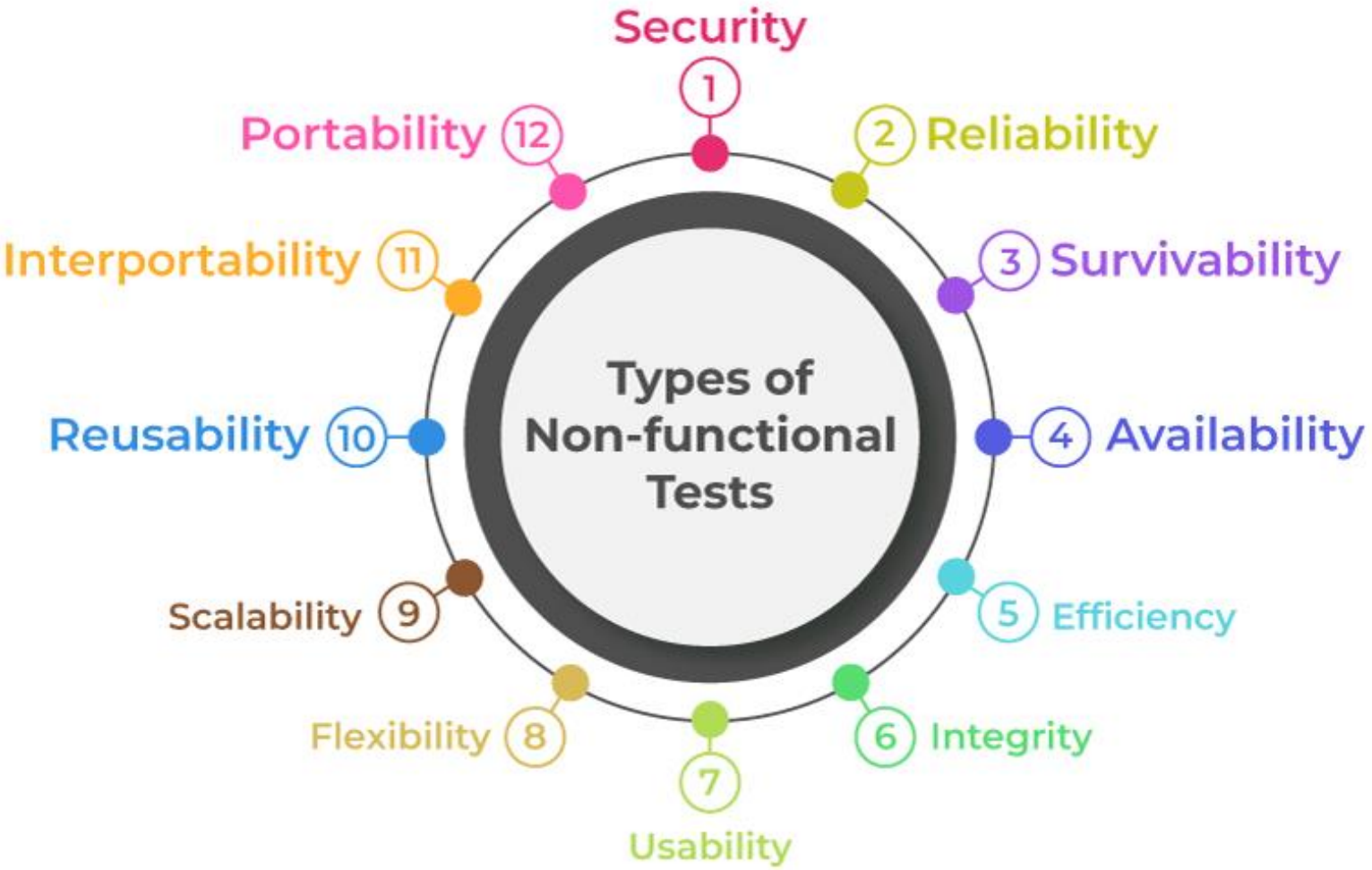
- **Defects can be identified early:** Software testing is important because if there are any bugs they can be identified early and can be fixed before the delivery of the software.
- **Improves quality of software:** Software Testing uncovers the defects in the software, and fixing them improves the quality of the software.
- **Increased customer satisfaction:** Software testing ensures reliability, security, and high performance which results in saving time, costs, and customer satisfaction.
- **Helps with scalability:** Software testing type non-functional testing helps to identify the scalability issues and the point where an application might stop working.
- **Saves time and money:** After the application is launched it will be very difficult to trace and resolve the issues, as performing this activity will incur more costs and time. Thus, it is better to conduct software testing at regular intervals during software development.

Benefits of Software Testing

- **Product quality:** Testing ensures the delivery of a high-quality product as errors are discovered and fixed early in the development cycle.
- **Customer satisfaction:** Software testing aims to detect the errors or vulnerabilities in the software early in the development phase so that the detected bugs can be fixed before the delivery of the product. Usability testing is a type of software testing that checks the application for how easily usable it is for the users to use the application.
- **Cost-effective:** Testing any project on time helps to save money and time for the long term. If the bugs are caught in the early phases of software testing, it costs less to fix those errors.
- **Security:** Security testing is a type of software testing that is focused on testing the application for security vulnerabilities from internal or external sources.


Type of Software Testing
Manual Testing
Automation Testing
White Box
Black Box
Grey Box
Functional Testing
Non-Functional Testing
Unit Testing
Integration Testing
System Testing
Performance Testing
Usability Testing
Compatibility Testing
Incremental Testing
Non-Incremental Testing
Top-down
Bottom-up
Load Testing
Stress Testing
Scalability Testing
Stability Testing


System Testing
Black Box
Regression Testing
Grey Box
Retesting
UAT
Smoke
Functional Testing Types
Component
Unit
Ad-hoc/Random
Integration
Acceptance
Database
Recovery
White Box
Static


Types of Non-functional Tests
1 Security
2 Reliability
3 Survivability
4 Availability
5 Efficiency
6 Integrity
7 Usability
8 Flexibility
9 Scalability
10 Reusability
11 Interportability
12 Portability

Dichotomies

The dichotomies in software testing to find the correct balance between two contrasting concepts to make your testing process effective. Within this field, we very often encounter dichotomies - which can lead to confusion when making decision. Dichotomies are crucial for a QA as it broaden their thought process and helps them in taking an informed decision during the design of test strategy.

Dichotomies in software testing

1. Manual Testing vs. Automated Testing

Manual testing involves human testers executing test cases, while automated testing involves using tools and scripts to execute test cases.

2. Black Box Testing vs. White Box Testing

Black box testing focuses on testing the functionality of the software without knowing its internal structure, while white box testing involves testing based on knowledge of the internal code and structure.

3. Functional Testing vs. Non-functional Testing

Functional testing checks if the software functions correctly according to its specifications, while non-functional testing evaluates aspects like performance, usability, security, etc.

4. Static Testing vs. Dynamic Testing

Static testing involves reviewing documentation, code, and requirements without executing the program, while dynamic testing involves executing the program's code.

Dichotomies in software testing

5. **Regression Testing vs. Retesting**

Regression testing ensures that changes to the software have not adversely affected existing functionality, while retesting focuses on verifying that defects have been fixed.

6. **Positive Testing vs. Negative Testing**

Positive testing checks if the system behaves as expected with valid inputs, while negative testing tests the system's ability to handle invalid inputs and unexpected behavior.

7. **Alpha Testing vs. Beta Testing**

Alpha testing is conducted by internal teams before the software is released to the public, while beta testing involves releasing the software to a limited group of external users for real-world testing.

8. **Sanity Testing vs. Smoke Testing**

Sanity testing checks whether the software is stable enough for further testing, while smoke testing verifies if the critical functionalities of the software work without encountering critical errors.

#	Testing	Debugging
1	Starts with known conditions. Uses predefined procedure. Has predictable outcomes.	Starts with possibly unknown initial conditions. End cannot be predicted.
2	Planned, Designed and Scheduled.	Procedures & Duration are not constrained.
3	A demo of an error or apparent correctness.	A Deductive process.
4	Proves programmer's success or failure.	It is programmer's Vindication.
5	Should be predictable, dull, constrained, rigid & inhuman.	There are intuitive leaps, conjectures, experimentation & freedom.

Functional Vs Structural Testing

- Functional Testing: Treats a program as a black box. Outputs are verified for conformance to specifications from user's point of view.
- Structural Testing: Looks at the implementation details: programming style, control method, source language, database & coding details.

#	Programmer / Designer	Tester
1	Tests designed by designers are more oriented towards structural testing and are limited to its limitations.	With knowledge about internal test design, the tester can eliminate useless tests, optimize & do an efficient test design.
2	Likely to be biased.	Tests designed by independent testers are bias-free.
3	Tries to do the job in simplest & cleanest way, trying to reduce the complexity.	Tester needs to suspicious, uncompromising, hostile and obsessed with destroying program.

#	Modularity	Efficiency
1	Smaller the component easier to understand.	Implies more number of components & hence more # of interfaces increase complexity & reduce efficiency (=> more bugs likely)
2	Small components/modules are repeatable independently with less rework (to check if a bug is fixed).	Higher efficiency at module level, when a bug occurs with small components.
3	Microscopic test cases need individual setups with data, systems & the software. Hence can have bugs.	More # of test cases implies higher possibility of bugs in test cases. Implies more rework and hence less efficiency with microscopic test cases
4	Easier to design large modules & smaller interfaces at a higher level.	Less complex & efficient. (Design may not be enough to understand and implement. It may have to be broken down to implementation level.)

Objectives of the Testing

- Detecting bugs as soon as feasible in any situation.
- Avoiding errors in a project's and product's final versions.
- Inspect to see whether the customer requirements criterion has been satisfied.
- Last but not least, the primary purpose of testing is to gauge the project and product level of quality.

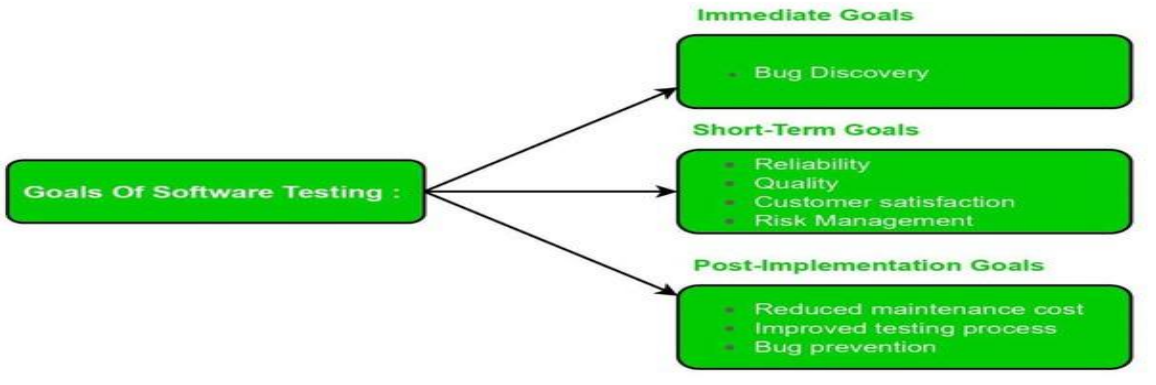


Fig : Software Testing Goals

Faults, Errors, Bugs, and Failures

Bug

A bug refers to defects, which means that the software product or application is not working as per the adhered requirements set. When we have any type of logical error, it causes our code to break, which results in a bug.

Defect

A defect refers to a situation when the application is not working as per the requirement and the actual and expected results of the application or software are not in sync with each other.

Error

It is a situation that happens when the development team or the developer fails to understand a requirement definition, and hence that misunderstanding gets translated into buggy code. This situation is referred to as an error and is mainly a term coined by the developers.

Fault

- Sometimes, due to certain factors, such as a lack of resources or not following proper steps, a fault occurs in software, which means that the logic was not incorporated to handle the errors in the application. This is an undesirable situation, but it mainly happens due to invalid documented steps or a lack of data definitions.

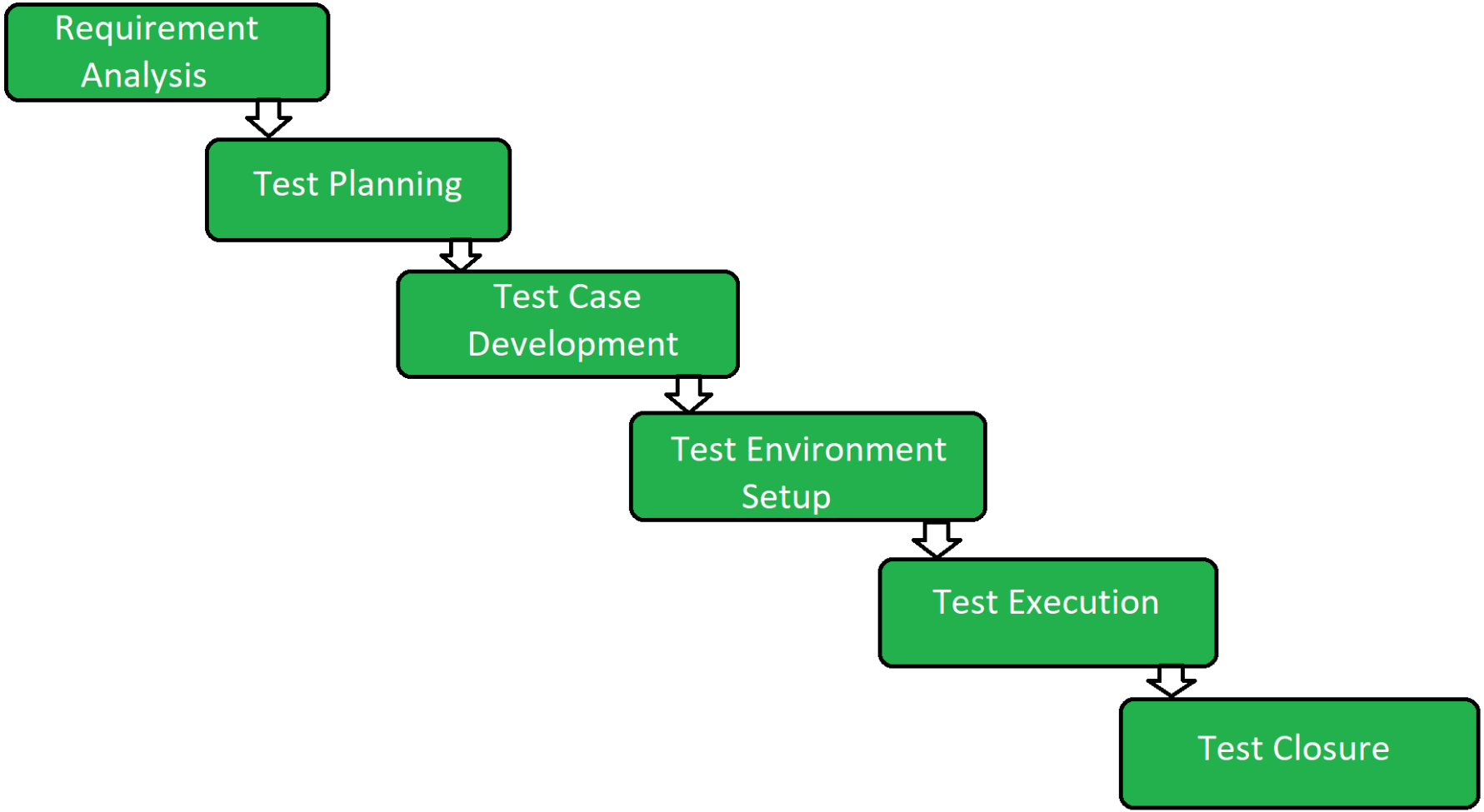
Failure

- Failure is the accumulation of several defects that ultimately lead to Software failure and results in the loss of information in critical modules, thereby making the system unresponsive. Generally, such situations happen very rarely because, before releasing a product, all possible scenarios and test cases for the code are simulated. Failure is detected by end-users once they face a particular issue in the software.


Errors/ Mistakes
Results in
Defects/ Bugs
Results in
Failures

STLC

- The Software Testing Life Cycle (STLC) is a systematic approach to testing a software application to ensure that it meets the requirements and is free of defects. It is a process that follows a series of steps or phases, and each phase has specific objectives and deliverables. The STLC is used to ensure that the software is of high quality, reliable, and meets the needs of the end-users.
- The main goal of the STLC is to identify and document any defects or issues in the software application as early as possible in the development process. This allows for issues to be addressed and resolved before the software is released to the public.
- The stages of the STLC include Test Planning, Test Analysis, Test Design, Test Environment Setup, Test Execution, Test Closure, and Defect Retesting. Each of these stages includes specific activities and deliverables that help to ensure that the software is thoroughly tested and meets the requirements of the end users.


Requirement Analysis
Test Planning
Test Case Development
Test Environment Setup
Test Execution
Test Closure

Test Cases

A Test Case is a specific set of conditions or variables under which a tester will determine whether an application, software system, or one of its features is working correctly or not. A Test Case aims to validate software features' accuracy, completeness, and reliability.

A test case usually includes the following details:

Testcase ID, description, Action taken, ER, AR, Status, Remarks

A successful test case should identify correct and incorrect results, and any unexpected results should be reported as a bug. Test cases provide the basis for exhaustively testing a system and thus help increase the reliability and quality of the system by uncovering mistakes or gaps in the system.

Best Practice for writing good Test Case.

When it comes to writing good test cases, certain best practices should be followed.

- First, it is important to identify the purpose of the test case and what exactly needs to be tested.
- Next, the test case should be written clearly and concisely, with step-by-step instructions for each action that needs to be taken. Also, it is important to consider all possible scenarios and edge cases to ensure thorough testing.
- Another important factor is maintaining organization and structure within your testing process by creating a logical flow of tests covering different aspects of the tested system.
- At last, it is always recommended to review and refine your test cases occasionally to maintain their quality over time.

Module Name:-	Login
Test Case ID	gfg_01
Tester Name	Geek
Test Case Description	To check login functionality of geeksforgeeks website.
Prerequisites:	1. Stable internet connection. 2. Browser(Chrome, Firefox, internet explorer, etc.)
Tester's Name	Geek
Enviromental Information:-	1. OS: Windows/Linux/Mac 2. System: Laptop/Desktop

Test Scenario	Checking that after entering the correct username and password, the user can login.

Test Case ID	Test Steps	Test Input	Expected Results	Actual Results	Status	Comments
1.	1. Enter Username	Username: geeksforgeeks	Welcome to geeksforgeeks!	Welcome to geeksforgeeks!	Pass	No issues found.
	2. Enter Password	Password: geek123				
	3. Click on login					

Quality and Reliability

Quality and reliability are related but distinct concepts in software testing.

- ✓ Quality refers to the degree to which a software product conforms to its specifications, requirements, and standards.
- ✓ Reliability refers to the probability that a software product will perform its intended functions without failure or error under specified conditions and time.
- ✓ Quality and reliability can be influenced by many factors, such as design, development, testing, maintenance, and user feedback.

Measuring quality

Various metrics can be used to measure software quality, depending on the objectives and criteria of a software project.

- Defect density, which is the number of defects or errors per unit of size such as lines of code, function points, or modules, is a common metric used to assess quality.
- Additionally, customer satisfaction can be determined by surveys, reviews, ratings, or feedback forms.
- A higher customer satisfaction indicates a higher quality level.
- Lastly, code coverage is the percentage of source code that is tested by a test suite or testing tool. A higher code coverage reduces the risk of undetected defects and increases the quality level.

Measuring reliability

Software reliability can be measured in various ways, depending on the type and complexity of the software product.

- For example, failure rate measures the frequency or probability of failures or errors occurring in a software product during a given period of time or usage, with a lower failure rate indicating a higher reliability level.
-
- Mean time between failures (MTBF) is another metric that indicates reliability, with a higher MTBF indicating a higher reliability level.

- Mean time to repair (MTTR) is the average time or effort required to fix or correct a failure or error in a software product, and a lower MTTR suggests higher reliability.

Applying metrics

- ✓ To apply software quality and reliability metrics effectively, it is important to define the scope and objectives of the software project, as well as the metrics to be used.

- ✓ You must then collect and analyze the data related to the software product and its metrics. Afterwards, compare the results with expected or desired values or standards.
- ✓ Additionally, identify and prioritize areas or aspects that need improvement or enhancement.
- ✓ Finally, implement and monitor changes that aim to improve or enhance the software quality and reliability.

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

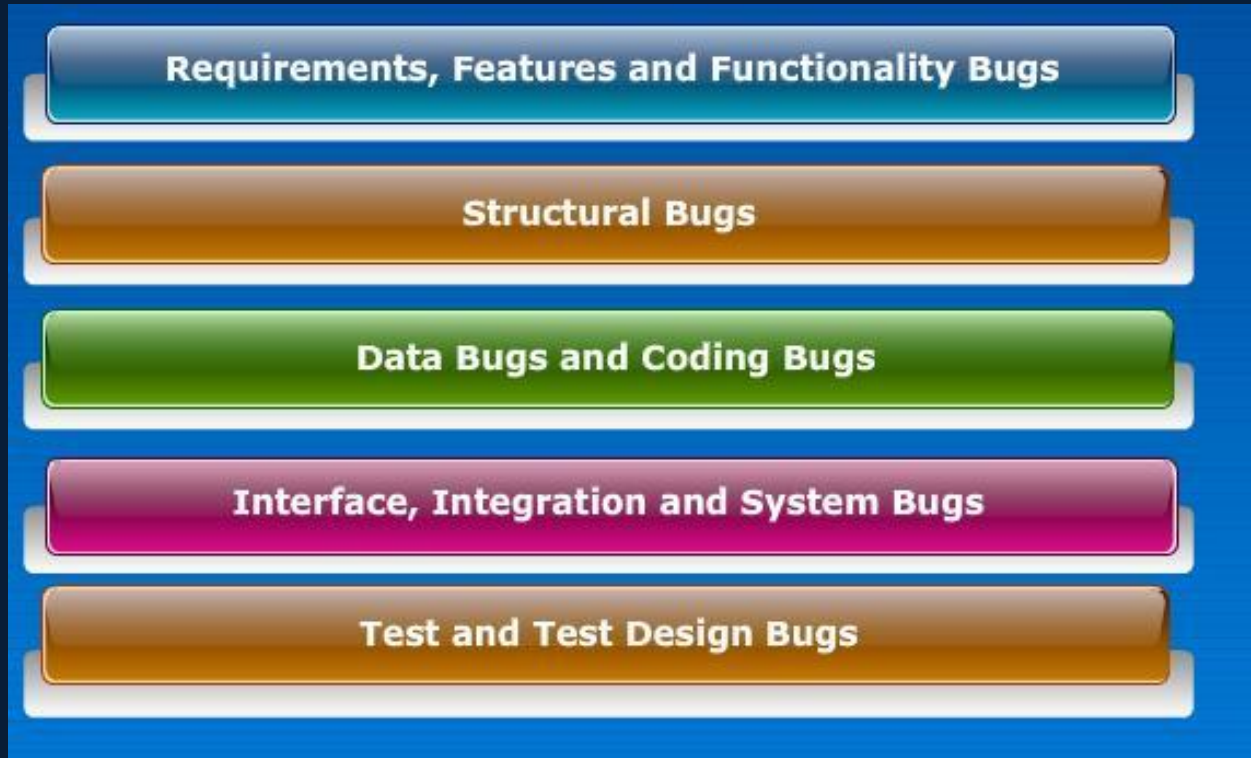
8. Intolerable: Long term unrecoverable corruption of the database occurs and the corruption is not easily discovered. Serious consideration is given to shutting the system down

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

Taxonomy of Bugs

There is no universally correct way to categorize bugs. The taxonomy is not rigid. A given bug can be put into one or another category depending on its history and the programmers state of mind.



Thank You

UNIT III (5 hours)

Introduction: Testing definition, Purpose of Testing, Dichotomies, Testing Objectives, Faults, Errors, Bugs, and Failures, Software Testing Life Cycle (STLC), Verification, Validation and Testing, Types of testing, Software Quality and Reliability, Software defect, Consequences of Bugs, Taxonomy of Bugs.

Testing definition:

A process of analyzing a software item to detect the differences between existing and required conditions (that is defects/errors/bugs) and to evaluate the features of the software item.

Who does Testing?

the following professionals are involved in testing a system within their respective capacities:

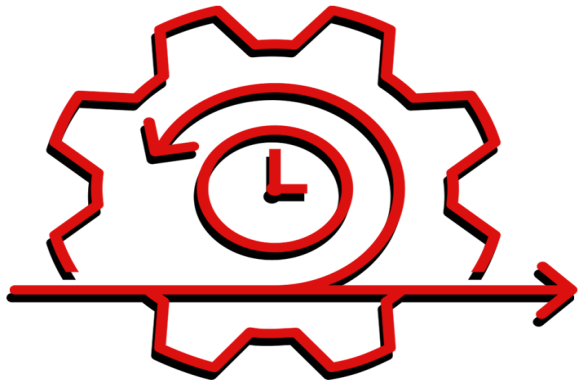
- ✓ Software Tester
- ✓ Software Developer
- ✓ Project Lead/Manager
- ✓ End User

Software testing can be divided into two steps:

1. **Verification:** . It means “Are we building the product right?”.
2. **Validation:** It means “Are we building the right product?”.

Building the
Right Product.

Building the
Product Right.



Need for Software Testing

- 1985: Canada's Therac-25 radiation therapy malfunctioned due to a software bug and resulted in lethal radiation doses to patients leaving 3 injured and 3 people dead.
- 1994: China Airlines Airbus A300 crashed due to a software bug killing 264 people.
- 1996: A software bug caused U.S. bank accounts of 823 customers to be credited with 920 million US dollars.
- 1999: A software bug caused the failure of a $1.2 billion military satellite launch.
- 2015: A software bug in fighter plan F-35 resulted in making it unable to detect targets correctly.
- 2015: Bloomberg terminal in London crashed due to a software bug affecting 300,000 traders on the financial market and forcing the government to postpone the 3bn pound debt sale.

Dichotomies in software testing

1. Manual Testing vs. Automated Testing

Manual testing involves human testers executing test cases, while automated testing involves using tools and scripts to execute test cases.

2. Black Box Testing vs. White Box Testing

Black box testing focuses on testing the functionality of the software without knowing its internal structure, while white box testing involves testing based on knowledge of the internal code and structure.

3. Functional Testing vs. Non-functional Testing

Functional testing checks if the software functions correctly according to its specifications, while non-functional testing evaluates aspects like performance, usability, security, etc.

4. Static Testing vs. Dynamic Testing

Static testing involves reviewing documentation, code, and requirements without executing the program, while dynamic testing involves executing the program's code.

Dichotomies in software testing

5. Regression Testing vs. Retesting

Regression testing ensures that changes to the software have not adversely affected existing functionality, while retesting focuses on verifying that defects have been fixed.

6.Positive Testing vs. Negative Testing

Positive testing checks if the system behaves as expected with valid inputs, while negative testing tests the system's ability to handle invalid inputs and unexpected behavior.

7. Alpha Testing vs. Beta Testing

Alpha testing is conducted by internal teams before the software is released to the public, while beta testing involves releasing the software to a limited group of external users for real-world testing.

8. Sanity Testing vs. Smoke Testing

Sanity testing checks whether the software is stable enough for further testing, while smoke testing verifies if the critical functionalities of the software work without encountering critical errors.

Objectives of Software Testing

The objectives of software testing are to:

1. Enhance software reliability and stability through rigorous testing.
2. Identify performance bottlenecks and optimize software efficiency.
3. Validate software compatibility across different platforms and environments.
4. Verify proper handling of edge cases and exceptional scenarios.
5. Detect and address security vulnerabilities to protect user data.
6. Ensure smooth integration with external systems or APIs.
7. Validate proper functioning of user interfaces and user experience.
8. Optimize software usability and accessibility for end-users.
9. Confirm compliance with industry standards and regulations.
10. Mitigate risks associated with software failures or malfunctions.

What is a Bug?

A bug refers to defects which means that the software product or the application is not working as per the adhered requirements set. When we have any type of logical error, it causes our code to break, which results in a bug. It is now that the Automation/ Manual Test Engineers describe this situation as a bug.

- A bug once detected can be reproduced with the help of standard bug-reporting templates.
- Major bugs are treated as prioritized and urgent especially when there is a risk of user dissatisfaction.
- The most common type of bug is a crash.
- Typos are also bugs that seem tiny but are capable of creating disastrous results.

What is a Defect?

A defect refers to a situation when the application is not working as per the requirement and the actual and expected result of the application or software are not in sync with each other.

- The defect is an issue in application coding that can affect the whole program.
- It represents the efficiency and inability of the application to meet the criteria and prevent the software from performing the desired work.
- The defect can arise when a developer makes major or minor mistakes during the development phase.

What is an Error?

Error is a situation that happens when the Development team or the developer fails to understand a requirement definition and hence that misunderstanding gets translated into buggy code. This situation is referred to as an Error and is mainly a term coined by the developers.

- Errors are generated due to wrong logic, syntax, or loop that can impact the end-user experience.
- It is calculated by differentiating between the expected results and the actual results.
- It raises due to several reasons like design issues, coding issues, or system specification issues and leads to issues in the application.

What is a Fault?

Sometimes due to certain factors such as Lack of resources or not following proper steps Fault occurs in software which means that the logic was not incorporated to handle the errors in the application. This is an undesirable situation, but it mainly happens due to invalid documented steps or a lack of data definitions.

- It is an unintended behavior by an application program.
- It causes a warning in the program.
- If a fault is left untreated it may lead to failure in the working of the deployed code.
- A minor fault in some cases may lead to high-end error.
- There are several ways to prevent faults like adopting programming techniques, development methodologies, peer review, and code analysis.

What is a Failure?

Failure is the accumulation of several defects that ultimately lead to Software failure and results in the loss of information in critical modules thereby making the system unresponsive. Generally, such situations happen very rarely because before releasing a product all possible scenarios and test cases for the code are simulated. Failure is detected by end-users once they face a particular issue in the software.

- Failure can happen due to human errors or can also be caused intentionally in the system by an individual.
- It is a term that comes after the production stage of the software.
- It can be identified in the application when the defective part is executed.

A simple diagram depicting Bug vs Defect vs Fault vs Failure:

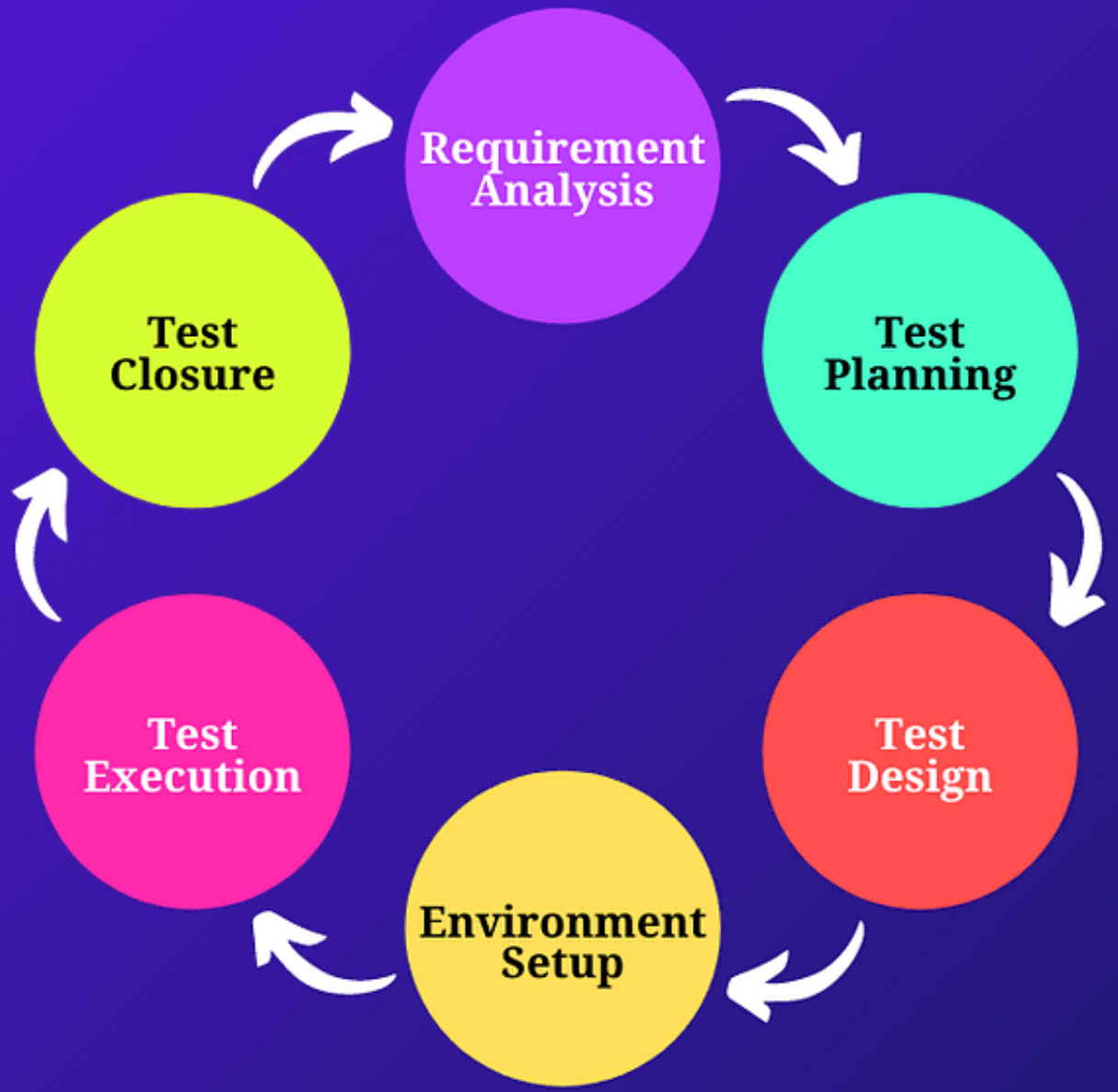

Errors/ Mistakes
Results in
Defects/ Bugs
Results in
Failures

Software Testing Life Cycle (STLC)

The Software Testing Life Cycle (STLC) is a systematic approach to testing a software application to ensure that it meets the requirements and is free of defects. It is a process that follows a series of steps or phases, and each phase has specific objectives and deliverables. The STLC is used to ensure that the software is of high quality, reliable, and meets the needs of the end-users.

The main goal of the STLC is to identify and document any defects or issues in the software application as early as possible in the development process. This allows for issues to be addressed and resolved before the software is released to the public.

The stages of the STLC include Test Planning, Test Analysis, Test Design, Test Environment Setup, Test Execution, Test Closure, and Defect Retesting. Each of these stages includes specific activities and deliverables that help to ensure that the software is thoroughly tested and meets the requirements of the end users.



SOFTWARE TESTING LIFE CYCLE

Phases of STLC

1. Requirement Analysis: Requirement Analysis is the first step of the Software Testing Life Cycle (STLC). In this phase quality assurance team understands the requirements like what is to be tested. If anything is missing or not understandable then the quality assurance team meets with the stakeholders to better understand the detailed knowledge of requirements.

2. Test Planning: Test Planning is the most efficient phase of the software testing life cycle where all testing plans are defined. In this phase manager of the testing, team calculates the estimated effort and cost for the testing work. This phase gets started once the requirement-gathering phase is completed.

3. Test Case Design: The test case development phase gets started once the test planning phase is completed. In this phase testing team notes down the detailed test cases. The testing team also prepares the required test data for the testing. When the test cases are prepared then they are reviewed by the quality assurance team.

4. Test Environment Setup: Test environment setup is a vital part of the STLC. Basically, the test environment decides the conditions on which software is tested. This is independent activity and can be started along with test case development. In this process, the testing team is not involved. either the developer or the customer creates the testing environment.

5. Test Execution: After the test case development and test environment setup test execution phase gets started. In this phase testing team starts executing test cases based on prepared test cases in the earlier step.

6. Test Closure: Test closure is the final stage of the Software Testing Life Cycle (STLC) where all testing-related activities are completed and documented. The main objective of the test closure stage is to ensure that all testing-related activities have been completed and that the software is ready for release.

At the end of the test closure stage, the testing team should have a clear understanding of the software's quality and reliability, and any defects or issues that were identified during testing should have been resolved. The test closure stage also includes documenting the testing process and any lessons learned so that they can be used to improve future testing processes

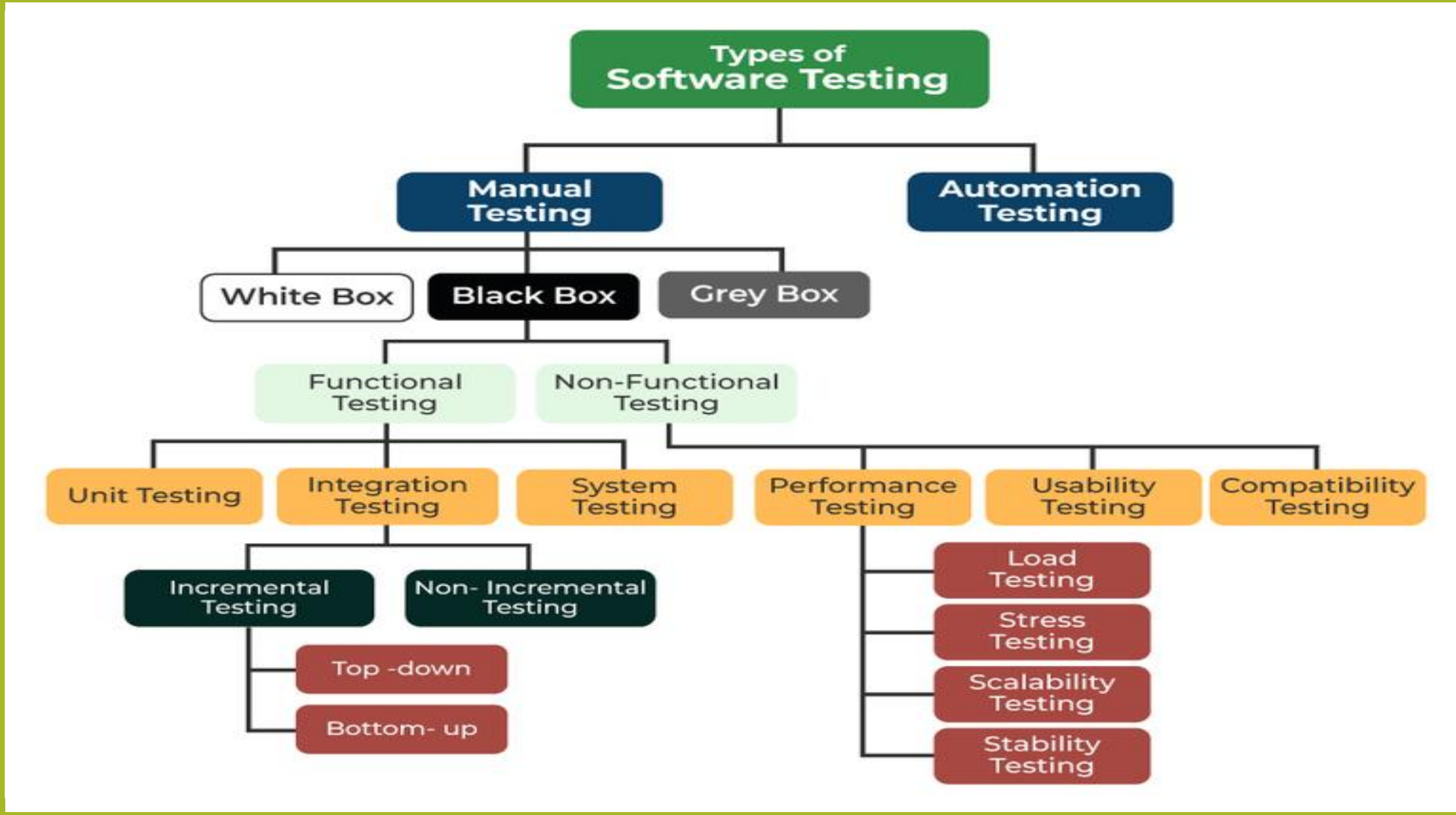
Characteristics of STLC

•STLC is a fundamental part of the Software Development Life Cycle (SDLC) but STLC consists of only the testing phases.

•STLC starts as soon as requirements are defined or software requirement document is shared by stakeholders.

•STLC yields a step-by-step process to ensure quality software.

Verification	Validation
We check whether we are developing the right product or not.	We check whether the developed product is right.
Verification is also known as **static testing**.	Validation is also known as **dynamic testing**.
Verification includes different methods like Inspections, Reviews, and Walkthroughs.	Validation includes testing like functional testing, system testing, integration, and User acceptance testing.
It is a process of checking the work-products (not the final product) of a development cycle to decide whether the product meets the specified requirements.	It is a process of checking the software during or at the end of the development cycle to decide whether the software follow the specified business requirements.
Quality assurance comes under verification testing.	**Quality control** comes under validation testing.
The execution of code does not happen in the verification testing.	In validation testing, the execution of code happens.
In verification testing, we can find the bugs early in the development phase of the product.	In the validation testing, we can find those bugs, which are not caught in the verification process.
Verification testing is executed by the Quality assurance team to make sure that the product is developed according to customers' requirements.	Validation testing is executed by the testing team to test the application.
Verification is done before the validation testing.	After verification testing, validation testing takes place.


Types of Software Testing
Manual Testing
Automation Testing
White Box
Black Box
Grey Box
Functional Testing
Non-Functional Testing
Unit Testing
Integration Testing
System Testing
Performance Testing
Usability Testing
Compatibility Testing
Incremental Testing
Non- Incremental Testing
Top -down
Bottom- up
Load Testing
Stress Testing
Scalability Testing
Stability 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.

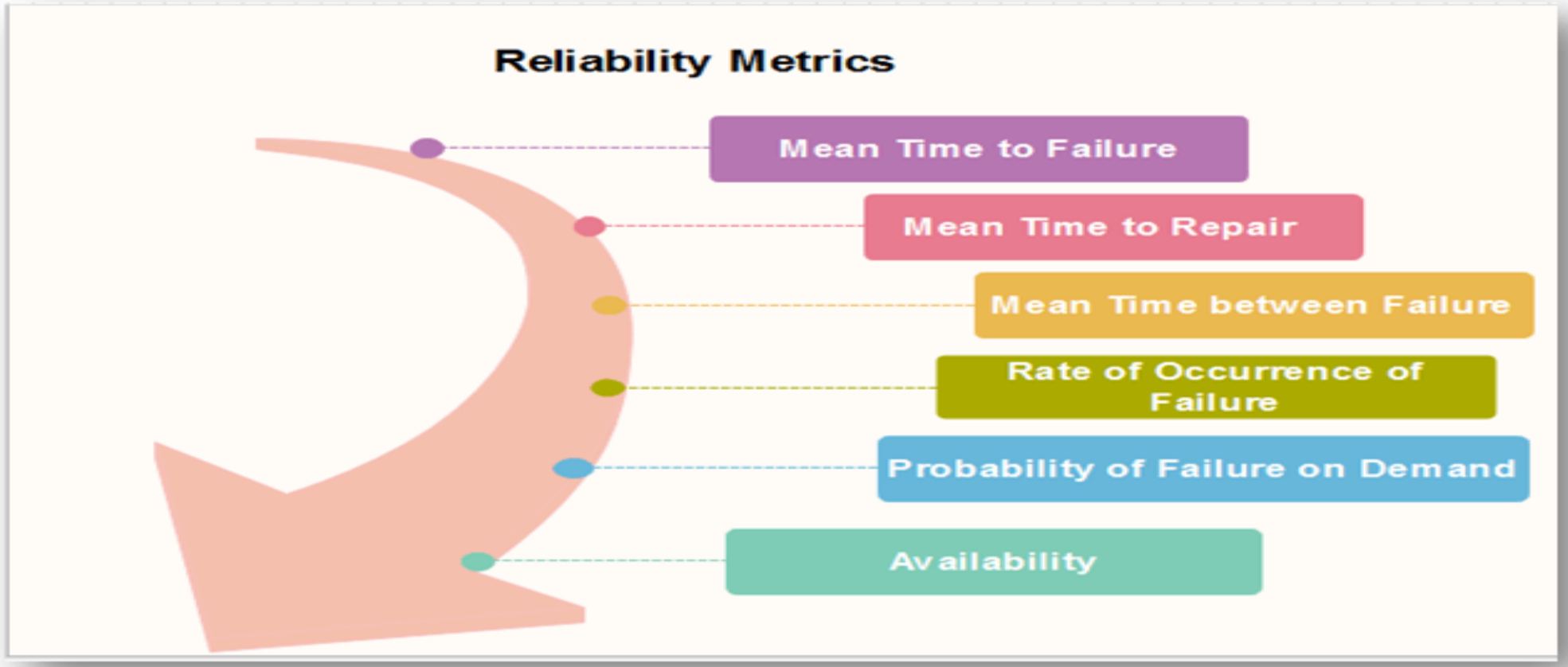
Advantages of Automation Testing:

- **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.

Reliability Metrics

Reliability metrics are used to quantitatively express the reliability of the software product. The option of which metric is to be used depends upon the type of system to which it applies & the requirements of the application domain.

Some reliability metrics which can be used to quantify the reliability of the software product are as follows:



1. Mean Time to Failure (MTTF)

MTTF is described as the time interval between the two successive failures. An **MTTF** of 200 mean that one failure can be expected each 200-time units. The time units are entirely dependent on the system & it can even be stated in the number of transactions. **MTTF** is consistent for systems with large transactions.
For example, It is suitable for computer-aided design systems where a designer will work on a design for several hours as well as for Word-processor systems.

2. Mean Time to Repair (MTTR)

Once failure occurs, some-time is required to fix the error. **MTTR** measures the average time it takes to track the errors causing the failure and to fix them.

3. Mean Time Between Failure (MTBR)

We can merge **MTTF** & **MTTR** metrics to get the MTBF metric.

MTBF = MTTF + MTTR

Thus, an **MTBF** of 300 denoted that once the failure appears, the next failure is expected to appear only after 300 hours. In this method, the time measurements are real-time & not the execution time as in **MTTF**.

4. **Rate of occurrence of failure (ROCOF)**

It is the number of failures appearing in a unit time interval. The number of unexpected events over a specific time of operation. **ROCOF** is the frequency of occurrence with which unexpected role is likely to appear. A **ROCOF** of 0.02 mean that two failures are likely to occur in each 100 operational time unit steps. It is also called the failure intensity metric.

5. **Probability of Failure on Demand (POFOD)**

POFOD is described as the probability that the system will fail when a service is requested. It is the number of system deficiencies given several systems inputs.

POFOD is the possibility that the system will fail when a service request is made.

A **POFOD** of 0.1 means that one out of ten service requests may fail. **POFOD** is an essential measure for safety-critical systems. POFOD is relevant for protection systems where services are demanded occasionally.

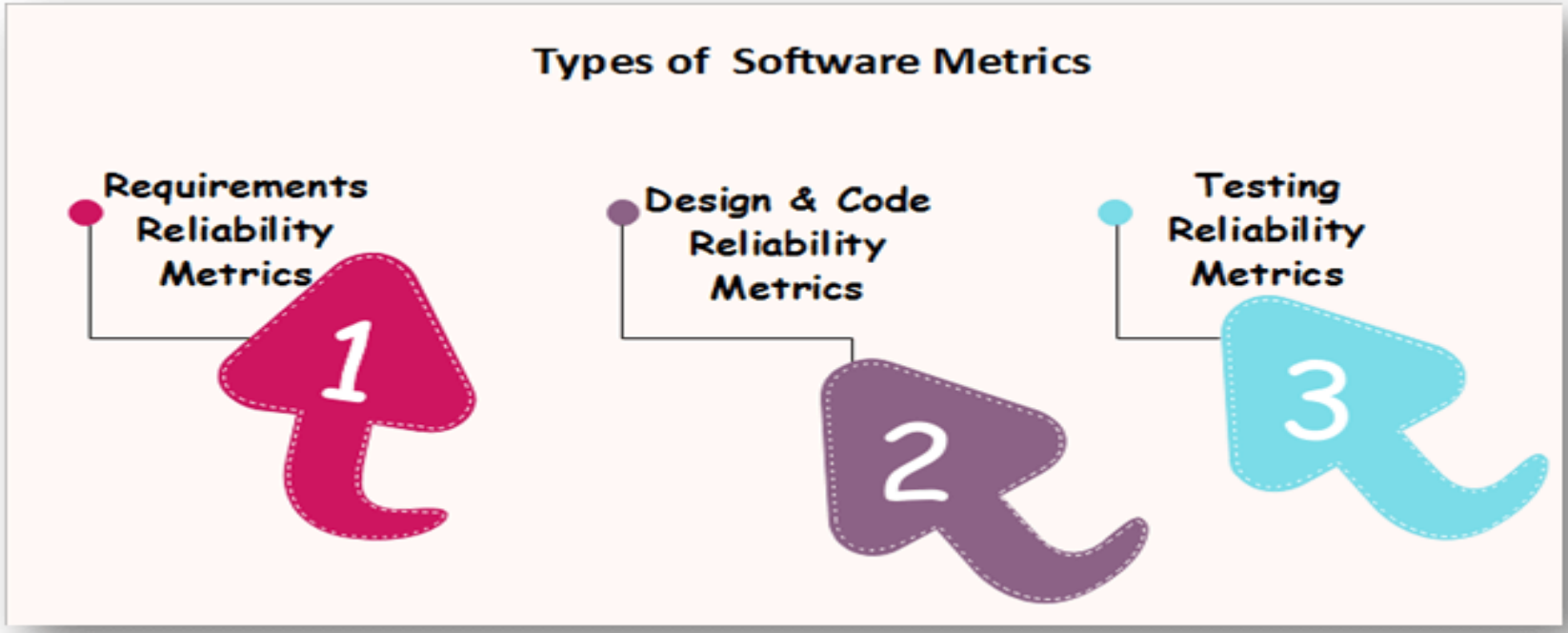
6. Availability (AVAIL)

Availability is the probability that the system is applicable for use at a given time. It takes into account the repair time & the restart time for the system. An availability of 0.995 means that in every 1000 time units, the system is feasible to be available for **995** of these. The percentage of time that a system is applicable for use, considering planned and unplanned downtime. If a system is down an average of four hours out of 100 hours of operation, its **AVAIL** is 96%.

Software Metrics for Reliability

The Metrics are used to improve the reliability of the system by identifying the areas of requirements.

Different Types of Software Metrics are:-



Requirements Reliability Metrics

Requirements denote what features the software must include. It specifies the functionality that must be contained in the software. The requirements must be written such that is no misconception between the developer & the client. The requirements must include valid structure to avoid the loss of valuable data. The requirements should be thorough and in a detailed manner so that it is simple for the design stage. The requirements should not include inadequate data. Requirement Reliability metrics calculates the above-said quality factors of the required document.

Design and Code Reliability Metrics

The quality methods that exists in design and coding plan are complexity, size, and modularity. Complex modules are tough to understand & there is a high probability of occurring bugs. The reliability will reduce if modules have a combination of high complexity and large size or high complexity and small size. These metrics are also available to object-oriented code, but in this, additional metrics are required to evaluate the quality.

Testing reliability metrics

These metrics are classified into two methods

First, it provides that the system is equipped with the tasks that are specified in the requirements. Because of this, the bugs due to the lack of functionality reduces.

The **second** method is calculating the code, finding the bugs & fixing them. To ensure that the system includes the functionality specified, test plans are written that include multiple test cases. Each test method is based on one system state and tests some tasks that are based on an associated set of requirements. The goals of an effective verification program is to ensure that each element is tested, the implication being that if the system passes the test, the requirement?s functionality is contained in the delivered system.

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.

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.

UNIT IV (5 hours)

Structural Testing (White Box Testing), Functional Testing (Black Box Testing), Integration Testing,
Regression Testing, Testing for Functionality and Testing for Performance, Top-Down and Bottom-Up,
Acceptance Testing, Alpha and Beta Testing of Products.

unit-IV

Structural Testing (White Box Testing)

white box testing which also known as glass box is **testing, structural testing, clear box testing, open box testing and transparent box testing**.

It tests the internal coding and infrastructure of a software focus on checking predefined inputs against expected and desired outputs.

It is based on the inner workings of an application and revolves around internal structure testing.

In this type of testing programming skills are required to design test cases.

The primary goal of white box testing is to focus on the flow of inputs and outputs through the software and strengthen the security of the software.

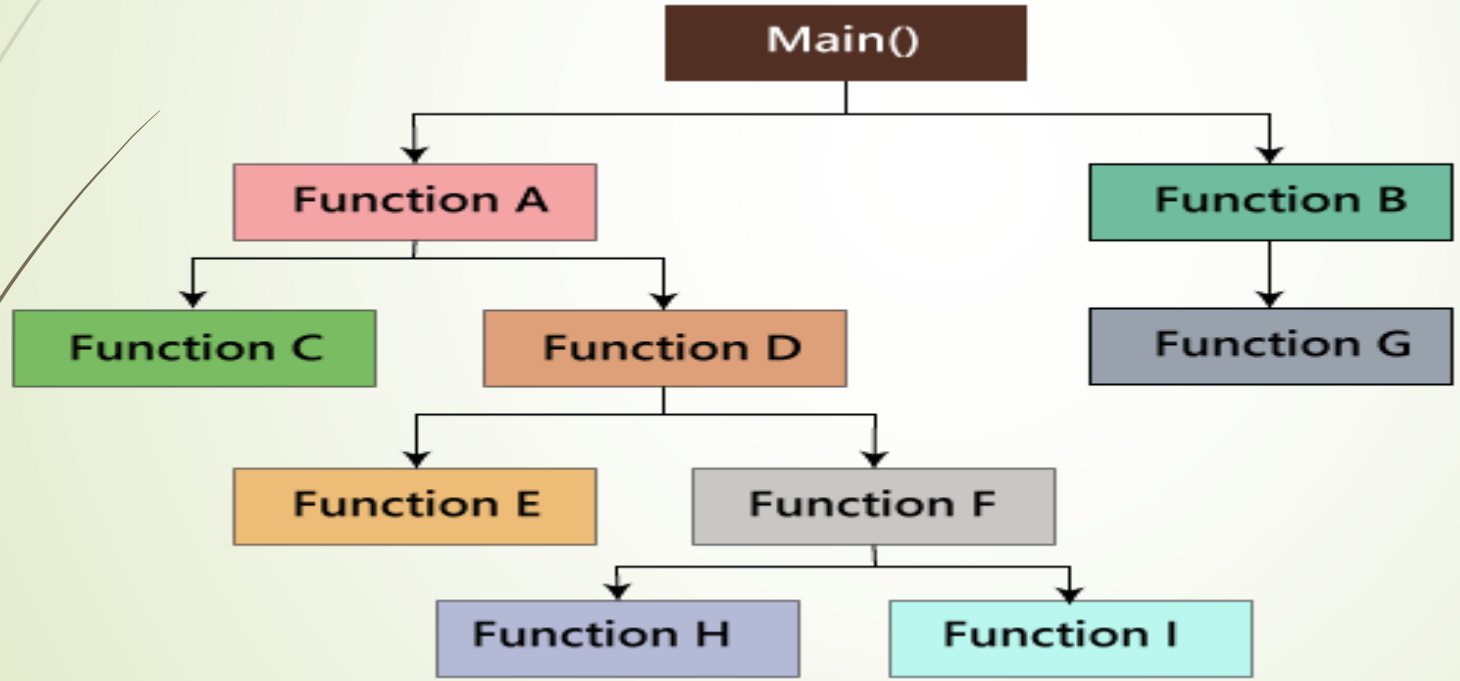
Developers do white box testing. In this, the developer will test every line of the code of the program.

The white box testing contains various tests, which are as follows:

- Path testing
- Loop testing
- Condition testing
- Testing based on the memory perspective
- Test performance of the program

Path testing

In the path testing, we will write the flow graphs and test all independent paths. Here writing the flow graph implies that flow graphs are representing the flow of the program and also show how every program is added with one another as we can see in the below image:



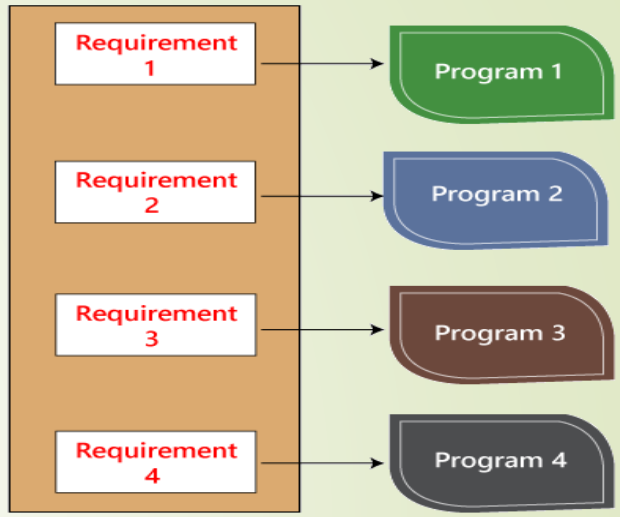
Loop testing

In the loop testing, we will test the loops such as while, for, and do-while, etc. and also check for ending condition if working correctly and if the size of the conditions is enough.

For example: we have one program where the developers have given about 50,000 loops.

1.{
2.while(50,000)
3.……
4.……
5.}

We cannot test this program manually for all the 50,000 loops cycle. So we write a small program that helps for all 50,000 cycles, as we can see in the below program, that test P is written in the similar language as the source code program, and this is known as a Unit test. And it is written by the developers only.



Condition testing

In this, we will test all logical conditions for both **true** and **false** values; that is, we will verify for both **if** and **else** condition.

For example:

```
if(condition) - true
{
.....
......
......
}
else - false
{
.....
......
......
}
```

The above program will work fine for both the conditions, which means that if the condition is accurate, and then else should be false and conversely.

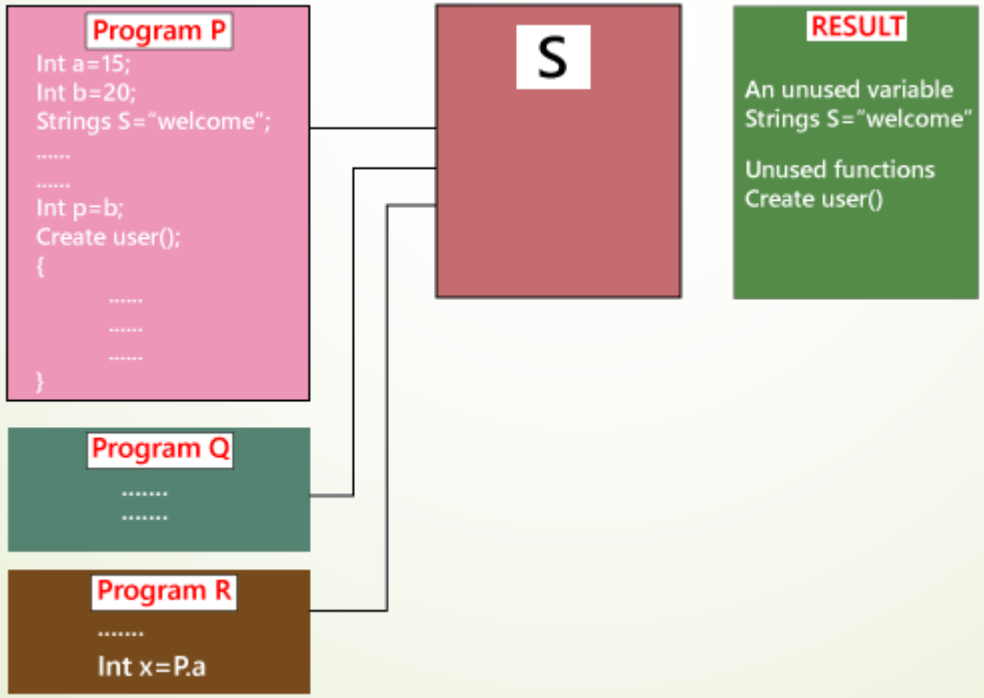
Testing based on the memory (size) perspective

The size of the code is increasing for the following reasons:

The reuse of code is not there: let us take one example, where we have four programs of the same application, and the first ten lines of the program are similar.

The **developers use the logic** that might be modified. If one programmer writes code and the file size is up to 250kb, then another programmer could write a similar code using the different logic, and the file size is up to 100kb.

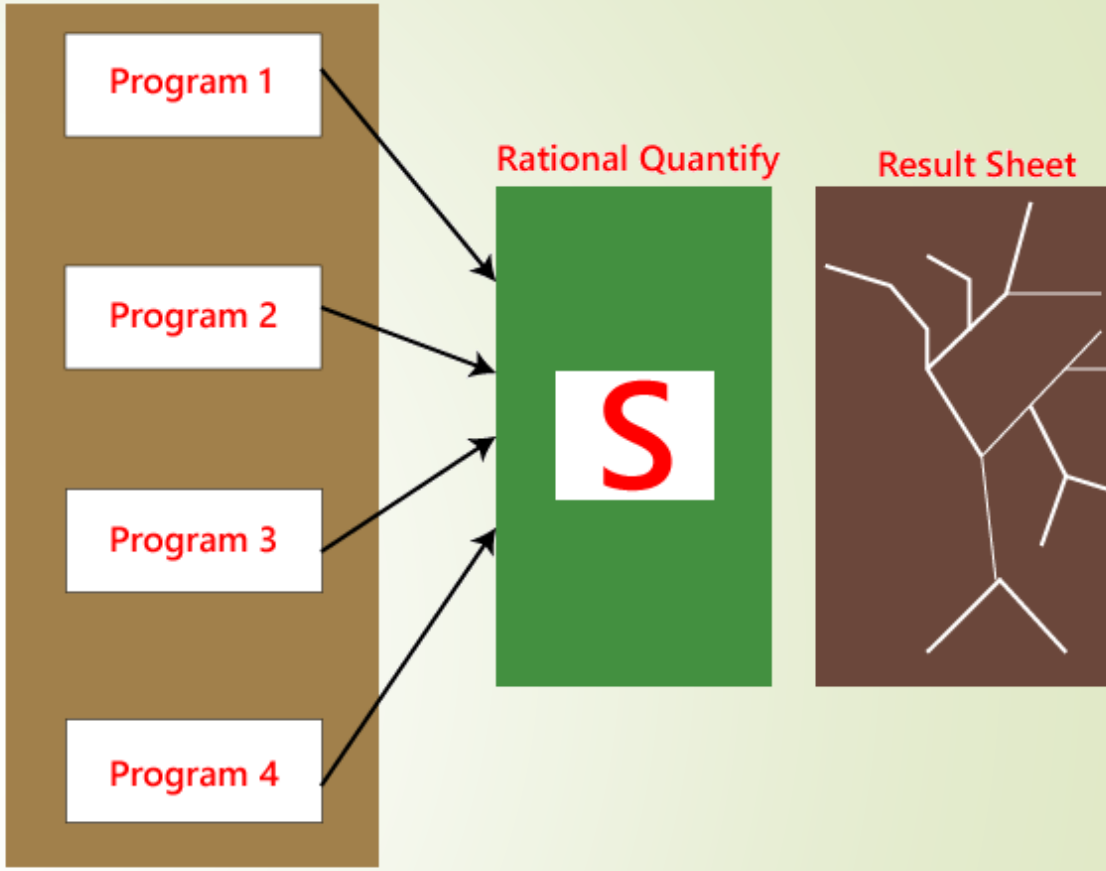
The **developer declares so many functions and variables** that might never be used in any portion of the code. Therefore, the size of the program will increase.



Test the performance (Speed, response time) of the program

The application could be slow for the following reasons:

- When logic is used.
- For the conditional cases, we will use **or** & **and** adequately.
- Switch case, which means we cannot use **nested if**, instead of using a switch case



Reasons for white box testing

- It identifies internal security holes.
- To check the way of input inside the code.
- Check the functionality of conditional loops.
- To test function, object, and statement at an individual level.

Advantages of White box testing

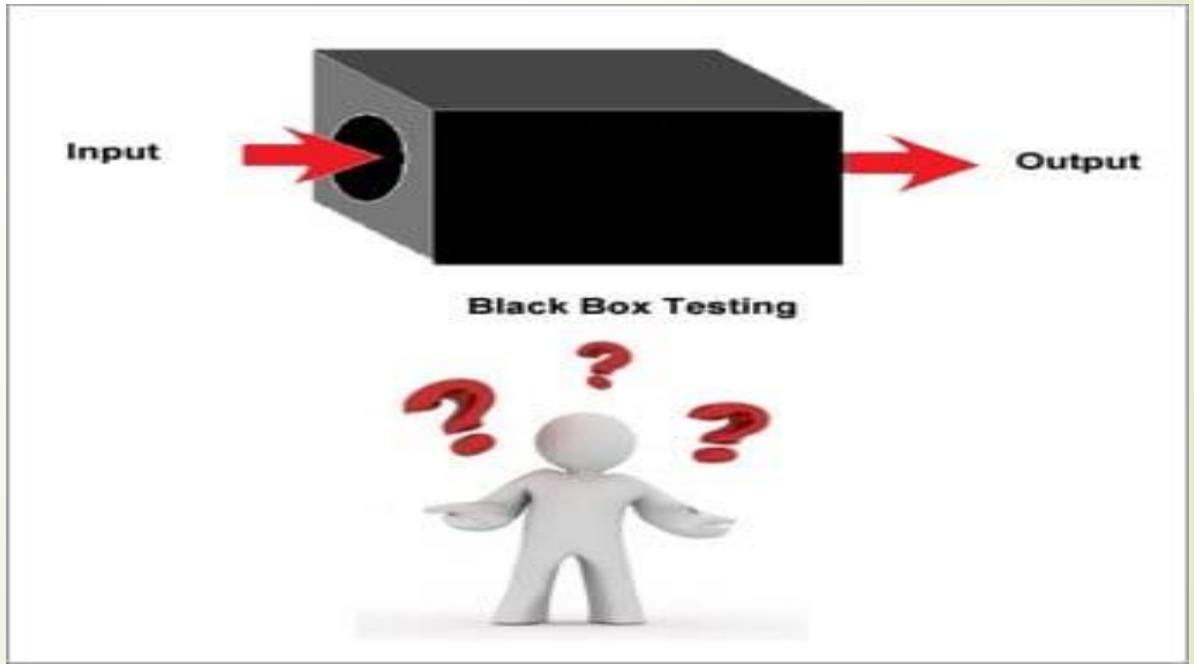
- White box testing optimizes code so hidden errors can be identified.
- Test cases of white box testing can be easily automated.
- This testing is more thorough than other testing approaches as it covers all code paths.
- It can be started in the SDLC phase even without GUI.

Disadvantages of White box testing

- White box testing is too time-consuming for large-scale programming applications.
- White box testing is much more expensive and complex.
- It can lead to production errors because the developers do not detail it.
- White box testing needs professional programmers with detailed knowledge and understanding of programming language and implementation.

What is Black Box Testing?

Black Box Testing is also known as behavioral, opaque(not transparent)-box, closed-box, specification-based or eye-to-eye testing.
It is a Software Testing method that analyzes the functionality of a software/application without knowing much about the internal structure/design of the item that is being tested and compares the input value with the output value.
The main focus of Black Box Testing is on the functionality of the system as a whole. The term **'Behavioral Testing'** is also used for Black Box Testing.
)



Types of Black Box Testing

1) Functional Testing

This testing type deals with the functional requirements or specifications of an application. Here, different actions or functions of the system are being tested by providing the input and comparing the actual output with the expected output.

For example, when we test a Dropdown list, we click on it and verify if it expands and all the expected values are showing in the list.

Few major types of Functional Testing are:

- Smoke Testing
- Sanity Testing
- Integration Testing
- System Testing
- Regression Testing
- User Acceptance Testing

2) Non-Functional Testing

Apart from the functionalities of the requirements, there are even several non-functional aspects that are required to be tested to improve the quality and performance of the application.

Few major types of Non-Functional Testing include:

- Usability Testing
- Load Testing
- Performance Testing
- Compatibility Testing
- Stress Testing
- Scalability Testing

Advantages

•The tester does not need to have a technical background. It is important to test by being in the user's shoes and thinking from the user's point of view.

•Testing can start once the development of the project/application is done. Both the testers and developers work independently without interfering in each other's space.

•It is more effective for large and complex applications.

•Defects and inconsistencies can be identified in the early stages of testing.

Disadvantages

•Without any technical or programming knowledge, there are chances of ignoring possible conditions of the scenario to be tested.

•In a stipulated time there is a possibility of testing less and skipping all possible inputs and their output testing.

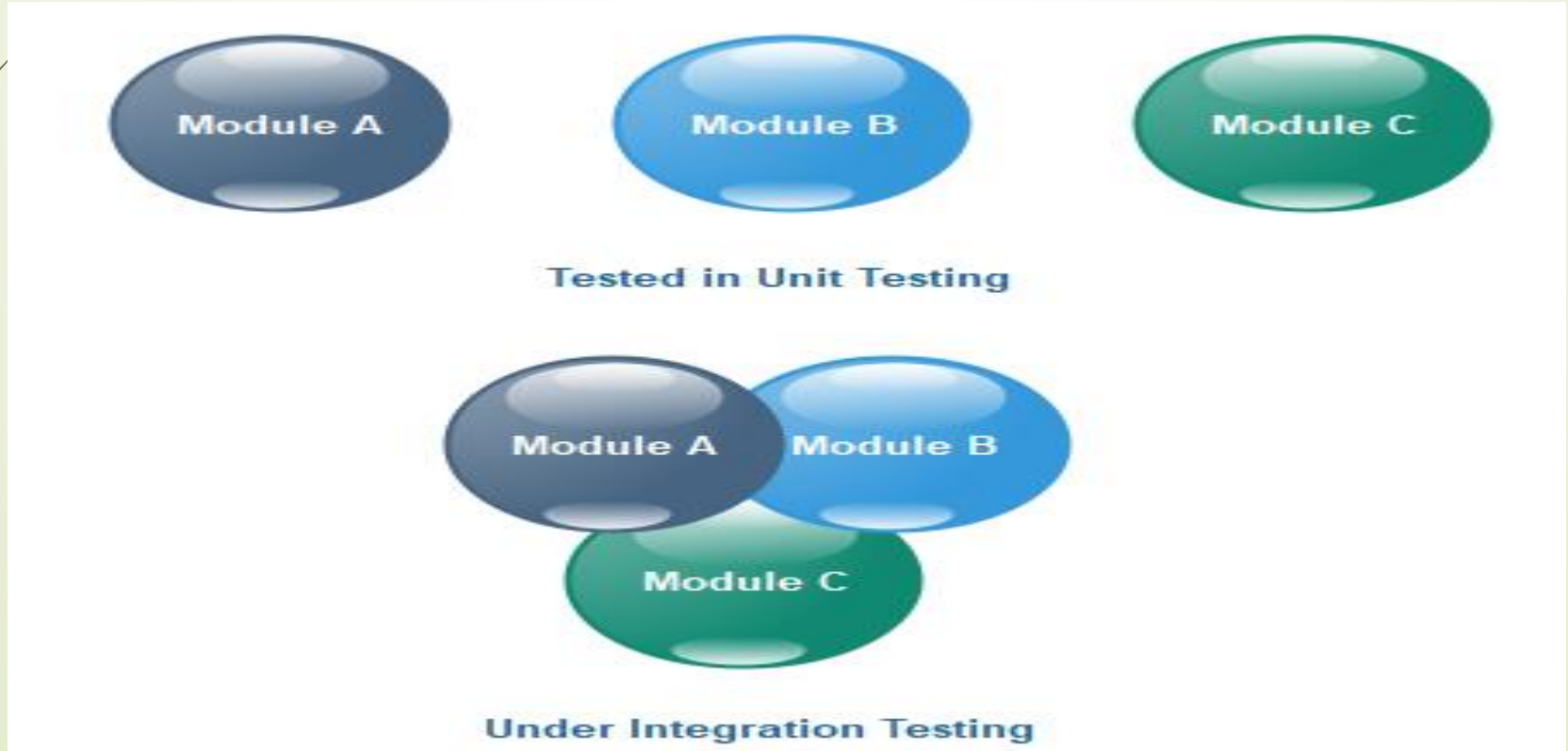
•Complete Test Coverage is not possible for large and complex projects.

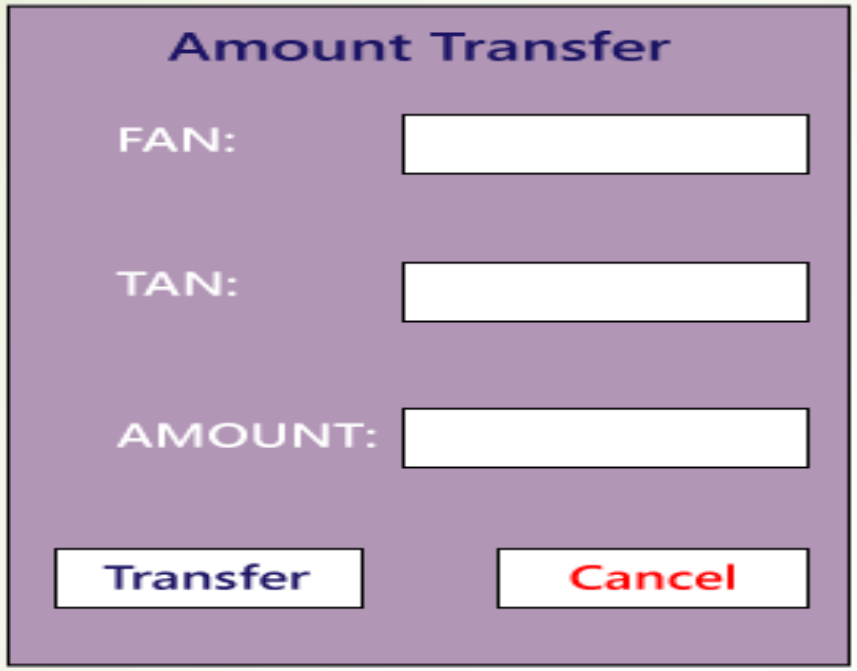
Difference Between White Box Testing and Black Box Testing

Black Box Testing	White Box Testing
It is a testing method without having knowledge about the actual code or internal structure of the application.	It is a testing method having knowledge about the actual code and internal structure of the application.
This is a higher level testing such as functional testing.	This type of testing is performed at a lower level of testing such as Unit Testing, Integration Testing.
It concentrates on the functionality of the system under test.	It concentrates on the actual code – program and its syntax's.
Black box testing requires Requirement specification to test.	White Box testing requires Design documents with data flow diagrams, flowcharts etc.
Black box testing is done by the testers.	White box testing is done by Developers or testers with programming knowledge.

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.





•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.

What is Regression Testing?

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 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.

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 application.**
2. 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.

When there is an environment change

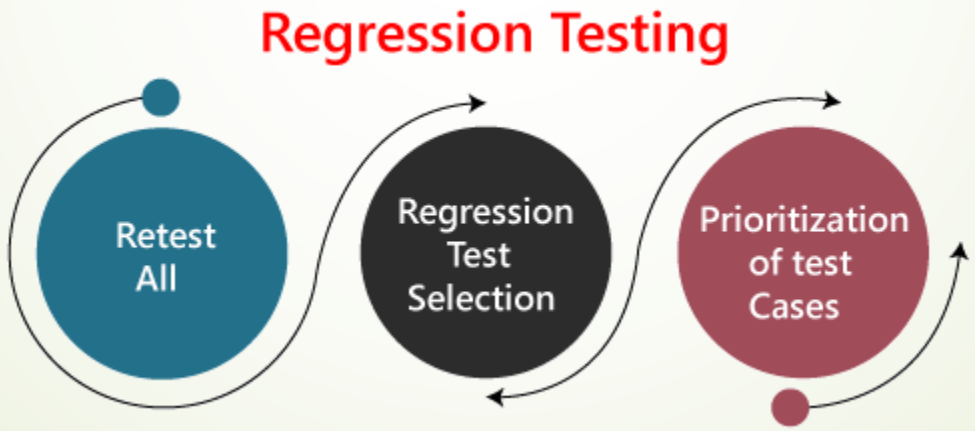
Example:

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

- Reusable Test cases.
- 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.

Alpha testing is a type of acceptance testing, which is performed to identify all possible bugs/issues before releasing the product to the end-user. Alpha test is a preliminary software field test carried out by a team of users to find out the bugs that were not found previously by other tests. Alpha testing is to simulate a real user environment by carrying out tasks and operations that actual user might perform. Alpha testing implies a meeting with a software vendor and client to ensure that the developers appropriately meet the client's requirements in terms of the performance, functionality, and durability of the software.

Beta Testing is a type of acceptance testing; it is the final test before shipping a product to the customers. Beta testing of a product is implemented by "real users "of the software application in a "real environment." In this phase of testing, the software is released to a limited number of end-users of the product to obtain feedback on the product quality. It allows the real customers an opportunity to provide inputs into the design, functionality, and usability of the product. These inputs are essential for the success of the product.

Differences between the Alpha testing and Beta testing are:

Sr. No.	Alpha Testing	Beta Testing
1.	Alpha testing performed by a team of highly skilled testers who are usually the internal employee of the organization.	Beta testing performed by clients or end-users in a real-time environment, who is not an employee of the organization.
2.	Alpha testing performed at the developer's site; it always needs a testing environment or lab environment.	Beta testing doesn't need any lab environment or the testing environment; it is performed at a client's location or end-user of the product.
3.	Reliability or security testing not performed in-depth in alpha testing.	Reliability, security, and robustness checked during beta testing.
4.	Alpha testing involves both white box and black-box techniques.	Beta testing uses only black-box testing.
5.	Long execution cycles maybe require for alpha testing.	Only a few weeks are required for the execution of beta testing.
6.	Critical issues or fixes can be identified by developers immediately in alpha testing.	Most of the issues or feedback is collecting from the beta testing will be implemented for the future versions of the product.
7.	Alpha testing performed before the launch of the product into the market.	At the time of software product marketing.
8.	Alpha testing focuses on the product's quality	Beta testing concentrates on the quality of the product,

UNIT V (5 hours)

Static Testing Strategies: Formal Technical Reviews (Peer Reviews), Walk Through, Regression testing,

Regression test process, Initial Smoke or Sanity test, Tools for regression testing,

Ad hoc Testing: Pair testing, Exploratory testing, Iterative testing, Defect seeding.

Test Planning, Management, Execution and Reporting, Software Test Automation: Testing in Object Oriented Systems.

Test Execution For Software Testing

The term Test Execution tells that the testing for the product or application needs to be executed in order to obtain the expected result. After the development phase, the testing phase will take place where the various levels of testing techniques will be carried out and the creation and execution of test cases will be taken place. The article focuses on discussing test execution.

Importance of Test Execution:

- **The project runs efficiently:** Test execution ensures that the project runs smoothly and efficiently.
- **Application competency:** It also helps to make sure the application's competency in the global market.
- **Requirements are correctly collected:** Test executions make sure that the requirements are collected correctly and incorporated correctly in design and architecture.
- **Application built in accordance with requirements:** It also checks whether the software application is built in accordance with the requirements or not.

Test Execution Process

The test Execution technique consists of three different phases which will be carried out to process the test result and ensure the correctness of the required results. Various team members will carry out various activities or work in each phase. The three main phases of test execution are the creation of test cases, test case execution, and validation of test results. Let us discuss each phase.

1. Creation of Test Cases: The first phase is to create suitable test cases for each module or function. Here, the tester with good domain knowledge must be required to create suitable test cases. It is always preferable to create simple test cases and the creation of test cases should not be delayed else it will cause excess time to release the product. The created test cases should not be repeated again. It should cover all the possible scenarios raised in the application.

2. Test Cases Execution: After test cases have been created, execution of test cases will take place. Here, the Quality Analyst team will either do automated or manual testing depending upon the test case scenario. It is always preferable to do both automated as well as manual testing to have 100% assurance of correctness. The selection of testing tools is also important to execute the test cases.

3. Validating Test Results: After executing the test cases, note down the results of each test case in a separate file or report. Check whether the executed test cases achieved the expected result and record the time required to complete each test case i.e., measure the performance of each test case. If any of the test cases is failed or not satisfied the condition then report it to the development team for validating the code.

Ways to Perform Test Execution

Testers can choose from the below list of preferred methods to carry out test execution:

1.Run test cases: It is a simple and easiest approach to run test cases on the local machine and it can be coupled with other artifacts like test plans, test suites, test environments, etc.

2.Run test suites: A test suite is a collection of manual and automated test cases and the test cases can be executed sequentially or in parallel. Sequential execution is useful in cases where the result of the last test case depends on the success of the current test case.

3.Run test case execution and test suite execution records: Recording test case execution and test suite execution is a key activity in the test process and helps to reduce errors, making the testing process more efficient.

4.Generate test results without execution: Generating test results from non-executed test cases can be helpful in achieving comprehensive test coverage.

5.Modify execution variables: Execution variables can be modified in the test scripts for particular test runs.

6.Run automated and manual tests: Test execution can be done manually or can be automated.

7.Schedule test artifacts: Test artifacts include video, screenshots, data reports, etc. These are very helpful as they document the results of the past test execution and provide information about what needs to be done in future test execution.

8.Defect tracking: Without defect tracking test execution is not possible, as during testing one should be able to track the defects and identify what when wrong and where.

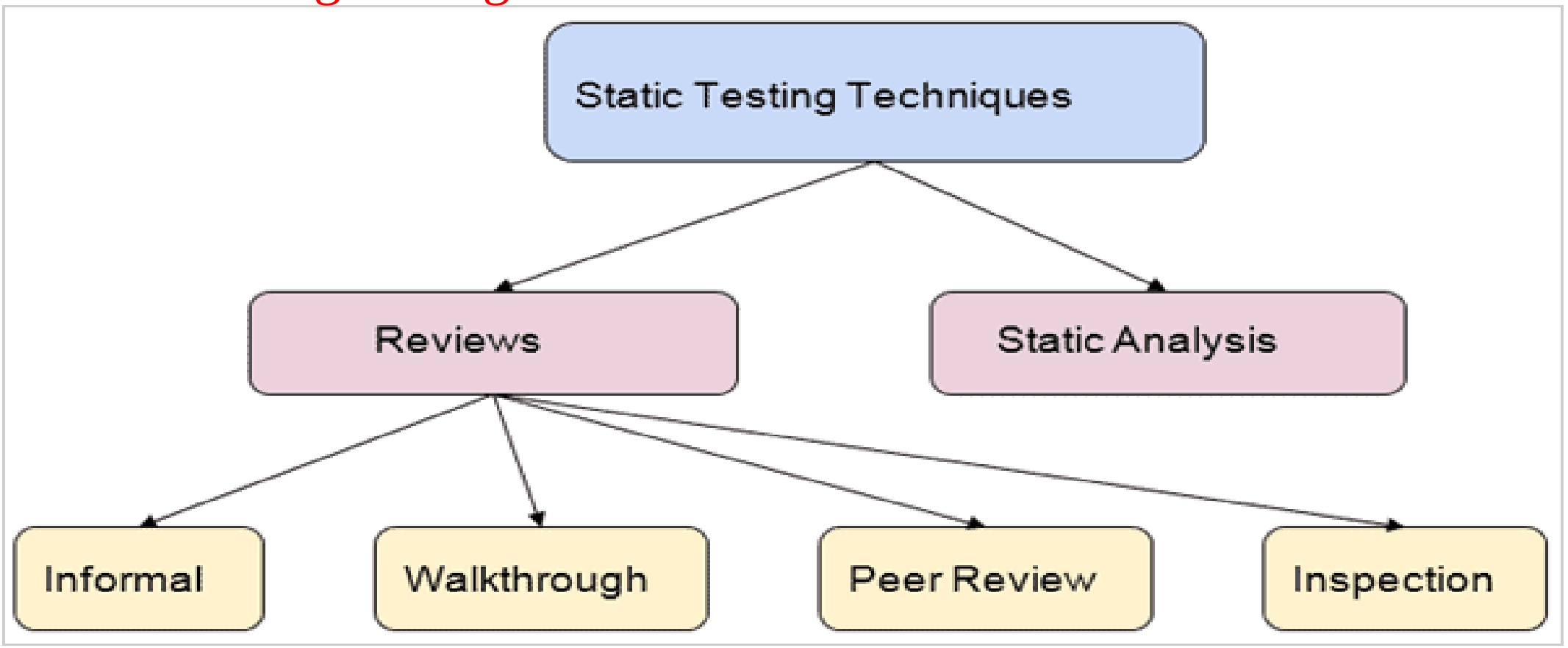
Introduction to Static Testing

Static testing is a verification process used to test the application without implementing the code of the application. And it is a **cost-effective process**.

To avoid the errors, we will execute Static testing in the initial stage of development because it is easier to identify the sources of errors, and it can fix easily.

In other words, we can say that **Static testing** can be done manually or with the help of tools to improve the quality of the application by finding the error at the early stage of development; that is also called the **verification process**. Static Testing Strategies:

Static Testing Strategies:



- **Informal reviews**

In **informal review**, the document designer place the contents in front of viewers, and everyone gives their view; therefore, bugs are acknowledged in the early stage.

- **Walkthrough**

Generally, the **walkthrough review** is used to performed by a skilled person or expert to verify the bugs. Therefore, there might not be problem in the development or testing phase.

- **Peer review**

In **Peer review**, we can check one another's documents to find and resolve the bugs, which is generally done in a team.

- **Inspection**

In review, the **inspection** is essentially verifying the document by the higher authority, **for example**, the **verification of SRS [software requirement specifications] document.**

Static Analysis

Another Static Testing technique is **static analysis**, which is used to contain the assessment of the code quality, which developers establish.

We can use different tools to perform the code's analysis and evaluation of the same.

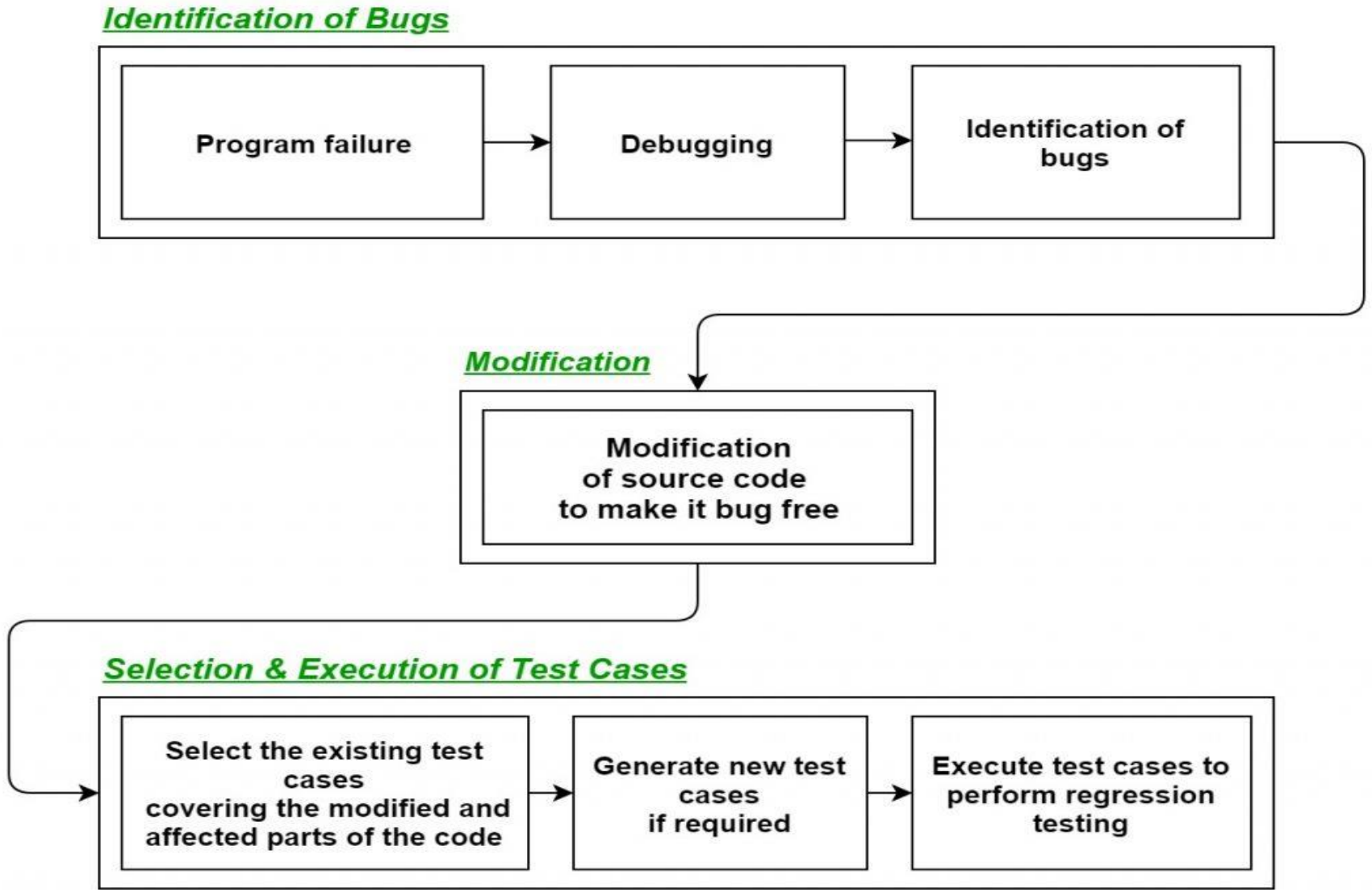
In other words, we can say that developers' developed code is analyzed with some tools for structural bugs, which might cause the defects.

The **static analysis** will also help us to identify the below errors:

- **Dead code**
- **Unused variables**
- **Endless loops**
- **Incorrect syntax**
- **Variable with undefined value**

Process of Regression testing

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


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

Techniques for the selection of Test cases for Regression Testing

- **Select all test cases:** In this technique, all the test cases are selected from the already existing test suite. It is the simplest and safest technique but not very efficient.
- **Select test cases randomly:** In this technique, test cases are selected randomly from the existing test suite, but it is only useful if all the test cases are equally good in their fault detection capability which is very rare. Hence, it is not used in most of the cases.
- **Select modification traversing test cases:** In this technique, only those test cases are selected that cover and test the modified portions of the source code and the parts that are affected by these modifications.
- **Select higher priority test cases:** In this technique, priority codes are assigned to each test case of the test suite based upon their bug detection capability, customer requirements, etc. After assigning the priority codes, test cases with the highest priorities are selected for the process of regression testing. The test case with the highest priority has the highest rank. For example, a test case with priority code 2 is less important than a test case with priority code 1.

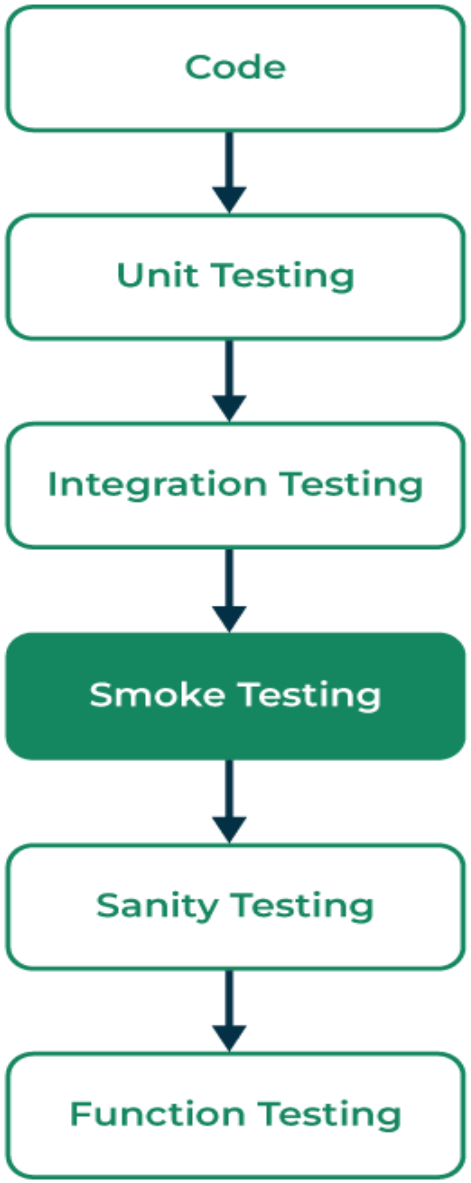
Tools for Regression testing

In regression testing, we generally select the test cases from the existing test suite itself and hence, we need not compute their expected output, and it can be easily automated due to this reason. Automating the process of regression testing will be very effective and time-saving. The most commonly used tools for regression testing are:

- **Selenium**
- **WATIR (Web Application Testing In Ruby)**
- **QTP (Quick Test Professional)**
- **RFT (Rational Functional Tester)**
- **Winrunner**
- **Silktest**

What is Smoke Testing?

Smoke Testing is a software testing method that determines whether the employed build is stable or not. It acts as a confirmation of whether the quality assurance team can proceed with further testing. Smoke tests are a minimum set of tests run on each build. Smoke testing is a process where the software build is deployed to a quality assurance environment and is verified to ensure the stability of the application. Smoke Testing is also known as ***Confidence Testing*** or ***Build Verification Testing***.



Characteristics of Smoke Testing:

The following are the characteristics of the smoke testing:

1.Level of Testing: Without delving into specific functionality, the testing procedure is superficial and broad-based, covering only the most important features.

2.Automation: Automated smoke tests are a common way to quickly and effectively confirm fundamental system functionality.

3.Frequency of execution: Usually, smoke testing is done following the release of a new build or following significant code modifications. In order to identify major issues early on, it can be run either daily or per build.

4.Time Management: The process of determining the build's stability is usually swift, requiring little time.

5.Environment: Usually, smoke tests are carried out in a controlled setting that is quite similar to the production setting.

ity is usually swift, requiring little time.

Environment: Usually, smoke tests are carried out in a controlled setting that is quite similar to the production setting.

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.
6.It saves test effort and time.
7.It makes it easy to detect critical errors and helps in the correction of errors.
8.It runs quickly.
9.It minimizes integration risks.

Disadvantages of Smoke Testing:

1.Smoke Testing does not cover all the functionality in the application. Only a certain part of the testing is done.
2.Errors may occur even after implementing all the smoke tests.
3.In the case of manual smoke testing, it takes a lot of time to execute the testing process for larger projects.
4.It will not be implemented against the negative tests or with the invalid input.
5.It usually consists of a minimum number of test cases and hence we cannot find the other issues that happened during the testing process.

What is 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.

Functionality of Sanity Testing:

- **Verification of Integration:** To make sure that recent adjustments or bug fixes haven't negatively impacted the integration of various modules or components, sanity testing may involve an integration check.
- **Verification of Fixed Bugs:** Sanity testing follows bug fixes or modifications to make that associated features continue to function appropriately and that reported issues have been satisfactorily resolved.
- **Efficiency of Time and Resources:** Sanity testing saves time and aids in resource optimization by rapidly determining whether a build is stable enough for further testing.
- **Check for Regression:** Sanity testing may include a rudimentary check for regressions to make sure that current functionalities have not been adversely affected, even if it is not as thorough as regression testing.
- **Repetitive Procedure:** It can be carried often frequently to swiftly validate each incremental build, particularly in agile and continuous integration setups.

Advantages of Sanity Testing:

•Sanity testing helps to quickly identify defects in the core functionality.
•It can be carried out in lesser time as no documentation is required for sanity testing.
•If the defects are found during sanity testing, project is rejected that is helpful in saving time for execution of regression tests.
•This testing technique is not so expensive when compared to another types of testing.
•It helps to identify the dependent missing objects.
•It is used to verify a small functionality of the system application whether it is still working or not even after a small change.
•It helps in the scenario when the time for testing of the product is limited or having less time to complete the test.

Disadvantages of Sanity Testing:

•It **focuses** only **on** the **functions** and **commands** of the system application.
•It is **not possible** to **cover all** the **test cases** in test scenarios.
•It **covers** only **few functionalities** in the system application. Issues in the unchecked functionalities can't be recovered.
•Sanity testing is usually **unscripted**. Hence, future references are not available.
•It **doesn't cover** the **design structure** level and hence it will be **difficult** for **development team** to identify and fix the issues.
•**Limited Scope:** The limited scope of sanity testing means that it may not uncover all potential issues or bugs in the software. This means that more comprehensive testing will be required to thoroughly validate the software's functionality.

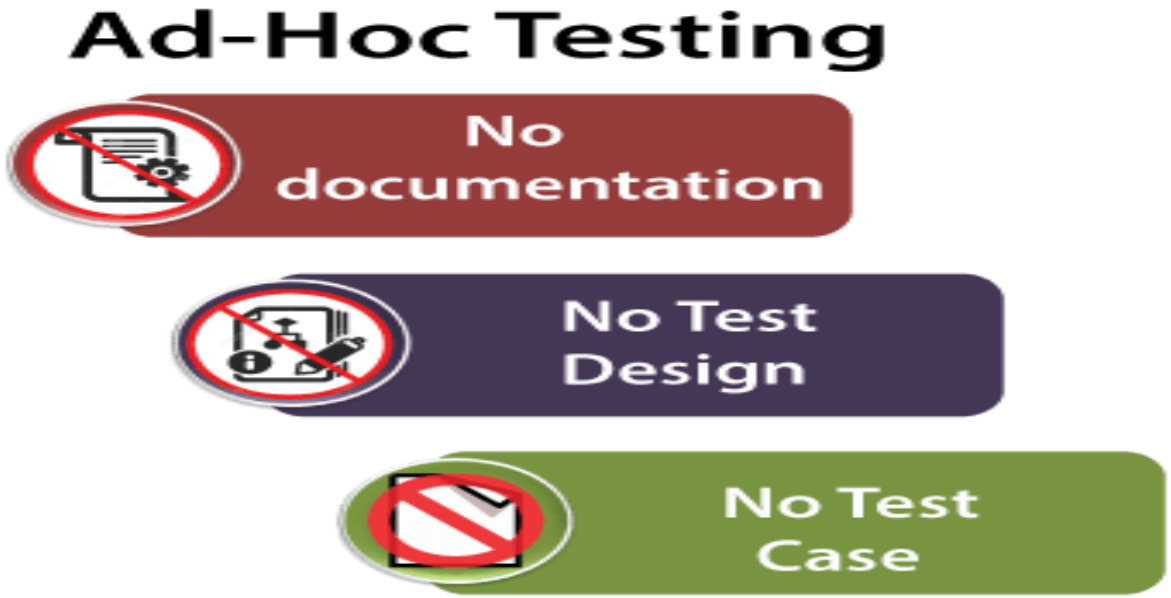
Adhoc Testing

This testing we do when the build is in the checked sequence, then we go for Adhoc testing by checking the application randomly.

Adhoc testing is also known as **Monkey testing and Gorilla testing**.

It is negative testing because we will test the application against the client's requirements.

When the end-user using the application randomly, and he/she may see a bug, but the professional test engineer uses the software systematically, so he/she may not find the same bug.



Suppose we are using two different browsers like Google Chrome and Mozilla Firefox and login to the Facebook application in both the browsers.
Then, we will change the password in the Google Chrome browser, and then in another browser (Firefox,) we will perform some action like sending a message.
It should navigate to the login page, and asking to fill the login credentials again because we change our credentials in another browser (Chrome), this process is called adhoc testing.

We go for Adhoc testing when all types of testing are performed. If the time permits then we will check all the negative scenarios during adhoc testing.

Advantages of Adhoc Testing

Following are some of the benefits of adhoc testing:

•Adhoc testing cannot follow any process; that why it can be performed anytime in the software development life cycle.

•The test engineer can test the application in their new ways that helps us to find out many numbers of bugs as compared to the actual testing process.

•The developer can also execute adhoc testing while developing the module that helps them to code in a better way.

•When the in-depth testing is required in less time, adhoc testing can be performed and also deliver the quality product on time.

•Adhoc testing does not require any documentation; that's why tester can test the application with more concentration without worrying about the formal documentation.

Disadvantages of Adhoc Testing

Following are the disadvantages of adhoc testing:

•Adhoc testing is dependent on the test engineer's product knowledge because he/she knows the flow of the application, so he/she knows where the application can collapse, and a new test engineer may not that much familiar with the application.

•Sometimes reproducing the bug is difficult because, in this testing, we did not follow any planning.

Pair Testing:

Pair Testing is verification in software by duo team members operating behind a machine. The first member controls the mouse and keyboards and the second member make notes discusses test scenarios and prepare/question. One of them has to be a tester and the next has to be a developer or business analyst.

Pair programming is a familiar practice in extreme programming. Therefore, Pair programming is considered a great approach to programming software. Likewise, pair testing is a similar process for testing software.

How to Perform Pair Testing in Software Testing:

Before starting the pair testing, some most important things to keep in mind are –

•**Pairing up with Right Person:** We can pair with anyone but it's better to pair if both individuals have some sense with each other's working process and goals effectively.

•**Allocating Proper Space:** The required pair need to allocate a device and space to seat together and perform the test properly. Basically, it will done via Video Conferencing tools where the driver would share the entire screen.

•**Establish the Goals:** After planning the structured approach to test, we need to keep an eye the areas to be covered, timebox the testing and aware about the required changes done.

•**Decide the Roles:** Before start the testing, we need to assign the role of driver and the navigator.

•**Logging bugs and taking notes:** The navigator need to take notes and maintain the bug log, when driver performs all the manual tasks. Once the process done, they should maintain a bug report and log all bugs.

Advantages Of Using Pair Testing:

1.Developers have an approach to the software from a different or positive view. They discuss and discover what happens if I execute it or what will happen if a business analyst doesn't implement it.

2.While Pair Testing is enforced with a business analyst, they exchange ideas and knowledge like an analyst and tester between them.

3.When a new project begins with the new team members there is often a hurdle between testers and developers.

4.If we find any trouble and want to register them in a bug registration system the trouble is automatically revisualized, therefore walking in pairs will help each other to be sharp.

Disadvantages Of Using Pair Testing:

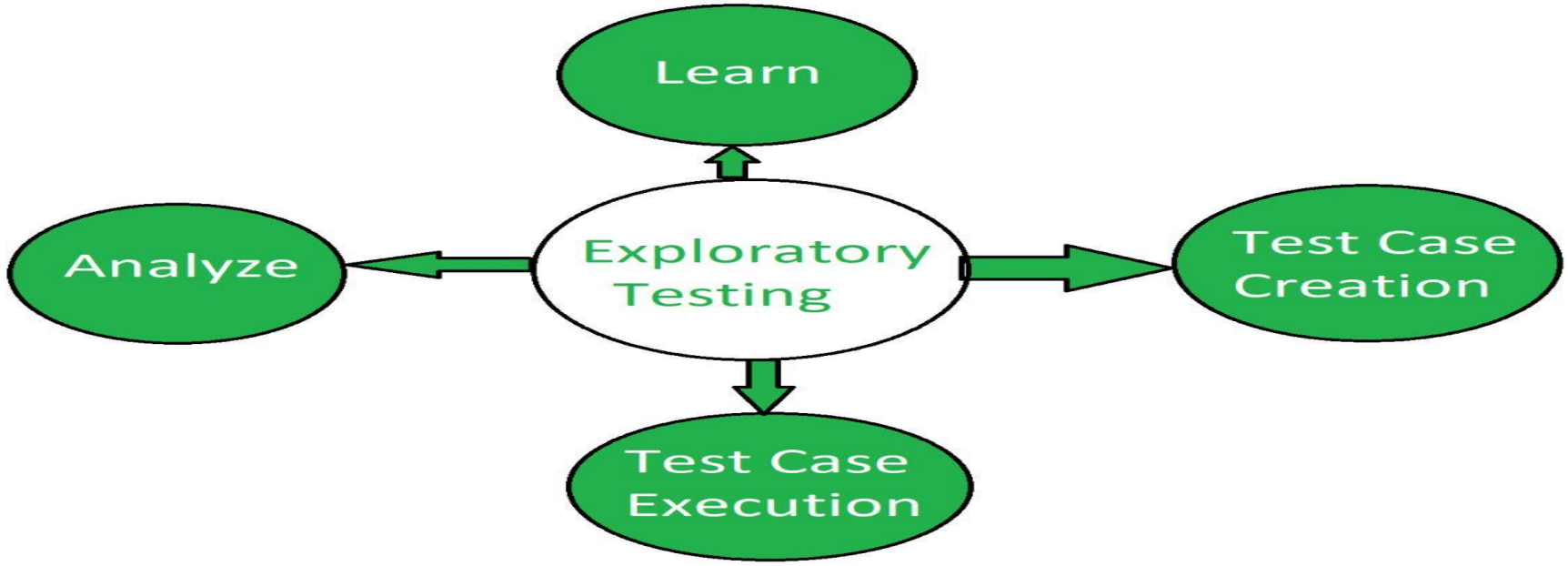
When you don't know the accurate condition for setting Pair Testing, you ought or not use it.

1.All enforced tests should be automatic and the solution of PT is findings and not test cases. We can't use the outcome of the PT session right after test automation.

2.When there are team members there are chances that two team members may end a classing with each other. That's why we should not use PT when the team members not communicating or working together well.

3.If you are planning to enforce structure test cases, it can't add more value or zero value executing the test cases together. The task should be performed by one team member alone.

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 identify the functional and technical faults in it. Exploratory testing aims to optimize and improve the software in every possible way. The exploratory testing technique combines the experience of testers with a structured approach to testing. It is often performed as a black box testing technique. 4 Exploratory testing is an unscripted testing technique.



1.Learn: This is the first phase of exploratory testing in which the tester learns about the faults or issues that occur in the software. The tester uses his/her knowledge, skill, and experience to observe and find what kind of problem the software is suffering from. This is the initial phase of exploratory testing. It also involves different new learning for the tester.
2.Test Case Creation: When the fault is identified i.e. tester comes to know what kind of problem the software is suffering from then the tester creates test cases according to defects to test the software. Test cases are designed by keeping in mind the problems end users can face.
3.Test Case Execution: After the creation of test cases according to end user problems, the tester executes the test cases. Execution of test cases is a prominent phase of any testing process. This includes the computational and operational tasks performed by the software to get the desired output.
4.Analysis: After the execution of the test cases, the result is analyzed and observed whether the software is working properly or not. If the defects are found then they are fixed and the above three steps are performed again. Hence this whole process goes on in a cycle and software testing is performed.

Below are some of the reasons for using exploratory testing:

- **Random and unstructured testing:** Exploratory testing is unstructured and thus can help to reveal bugs that would of undiscovered during structured phases of testing.
- **Testers can play around with user stories:** With exploratory testing, testers can annotate defects, add assertions, and voice memos and in this way, the user story is converted to a test case.
- **Facilitate agile workflow:** Exploratory testing helps formalize the findings and document them automatically. Everyone can participate in exploratory testing with the help of visual feedback thus enabling the team to adapt to changes quickly and facilitating agile workflow.
- **Reinforce traditional testing process:** Using tools for automated test case documentation testers can convert exploratory testing sequences into functional test scripts.
- **Speeds up documentation:** Exploratory testing speeds up documentation and creates an instant feedback loop.
- **Export documentation to test cases:** Integration exploratory testing with tools like Jira recorded documentation can be directly exported to test cases.

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.
- **Generation of new ideas:** Exploratory testing encourages creativity and intuition thus the generation of new ideas during test execution.
- **Catch defects missed in test cases:** Exploratory testing helps to uncover bugs that are normally ignored by other testing techniques.

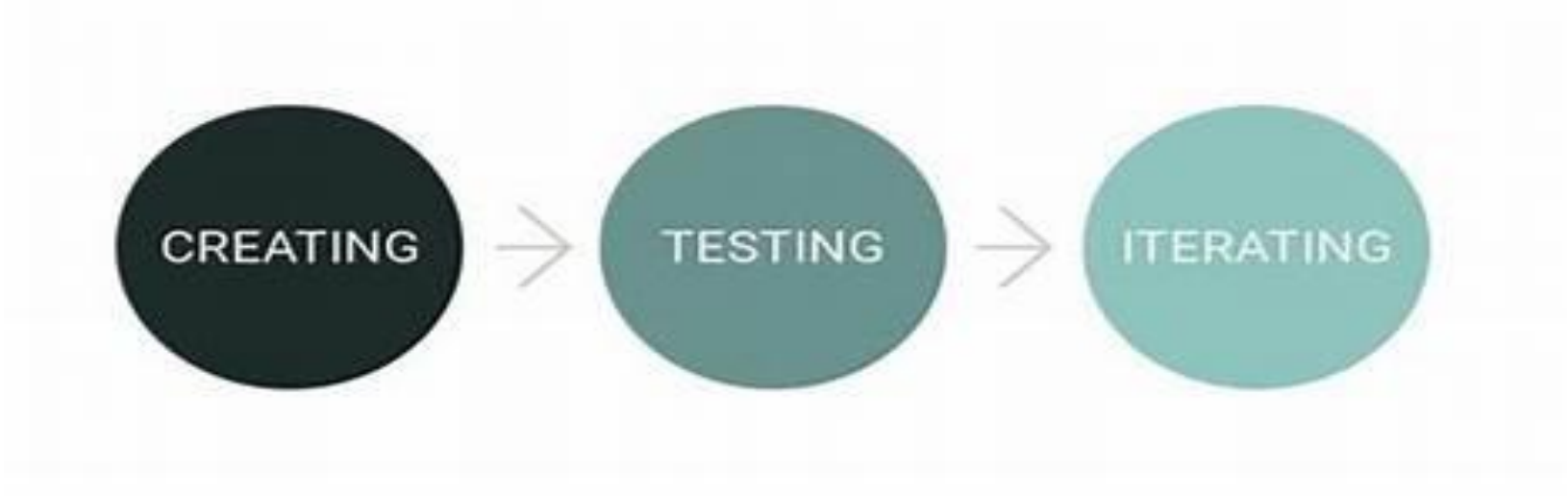
Disadvantages of Exploratory Testing:

- **Tests cannot be reviewed in advance:** In exploratory testing, Testing is performed randomly so once testing is performed it cannot be reviewed.
- **Dependent on the tester's knowledge:** In exploratory testing, the testing is dependent on the tester's knowledge, experience, and skill. Thus, it is limited by the tester's domain knowledge.
- **Difficult to keep track of tests:** In Exploratory testing, as testing is done in an ad-hoc manner, keeping track of tests performed is difficult.
- **Not possible to repeat test methodology:** Due to the ad-hoc nature of testing in exploratory testing, tests are done randomly and thus it is not suitable for longer execution time, and it is not possible to repeat the same test methodology.

Conclusion

What Is Iterative Testing?

Iterative testing refers to making small, gradual changes or updates to a product based on insights (e.g., test results and user feedback) from previous changes and testing them against predefined baseline metrics. It is commonly practiced in a UI/UX context but can be used in the context of product management.



Here are a few of the benefits of iterative testing for product managers:

1. Manage and Test Easily

Iterative testing enables product teams to make incremental, evidence-based changes to a feature or product. It allows them to roll changes out quickly and then gather user feedback to shape product decisions. Because the changes aren' sweeping ones, they are easier to manage and test.

2. Identify Issues Early

Gradual tweaks made to the product help product teams identify and eliminate bugs or usability issues and correct the early on. Getting ahead enables an organization to deliver a better product to users.

3. Get Better Insight

Iterative testing gives product managers actionable insights via test results and user feedback, which they can use to improve the product.

4. Deliver a Better Product

Product managers want to achieve product excellence by developing a significant or impactful product or feature and getting it to market quickly. Iterative testing helps product managers get to the heart of how users will engage with a product

5. Maintain Flexibility

Making small, gradual changes to a product helps product managers adapt to users' changing needs. They can keep close tabs on how users react to and feel about those changes. These valuable insights, in turn, help guide future product decisions.

6. Get Stakeholder Buy-In

Because iterative testing is evidence-based (i.e., real data and user feedback), product decisions are more comfortable to justify. This ultimately helps product managers make their cases with stakeholders

Defect Seeding

Defect Seeding is a technique that was developed to estimate the number of bugs resident in a piece of software.
Conceptually, a piece of software is "seeded" with bugs and then the test set is run to find out how many of the seeded bugs were discovered, how many were not discovered, and how many new (unseeded) bugs were found. It's then possible to use a simple mathematical formula to predict the number of bugs remaining.

If an organization inserted 100 seed bugs and later were only able to locate 75 of the seeded bugs, their seed ratio would be 0.75 (or 75%). If the organization had already discovered 450 "real" defects, then using the results from the seeding experiment, it would be possible to extrapolate that the 450 "real" defects represented only 75% of all of the real defects present. Then, the total number of real defects would be estimated to be 600. Since only 450 of the potential 600 real defects have been found, it appears that the product still has 150 "real" defects waiting to be discovered plus 25 seed bugs that still exist in the code. Don't forget to remove the seed bugs!

Estimated Number of Real Defects Still Present	=	Estimated Total Number of Real Defects	-	Number of Real Defects Found

Seed Ratio = Number of Seed Bugs Found / Total Number of Seed Bugs

Estimated Total Number of Real Defects	=	Number of Real Defects Found Seed Ratio

Number of Seed Bugs Found = 75

Total Number of Seed Bugs = 100

Seed Ratio = 75 / 100 = 0.75 or 75%

Estimated Total Number Of Real Defects	=	450/.75 = 600

Number of Real Defects Found = 450

Estimated Number Of Real Defects Still Present = 600 - 450 = 150

Test Plan

A **Test Plan** is a detailed document that describes the test strategy, objectives, schedule, estimation, deliverables, and resources required to perform testing for a software product. Test Plan helps us determine the effort needed to validate the quality of the application under test. The test plan serves as a blueprint to conduct software testing activities as a defined process, which is minutely monitored and controlled by the test manager.

As per ISTQB definition: “Test Plan is A document describing the scope, approach, resources, and schedule of intended test activities.”

Let’s start with following Test Plan example/scenario: In a meeting, you want to discuss the Test Plan with the team members, but they are not interested – .

What is the Importance of Test Plan?

Making Test Plan document has multiple benefits

•Help people outside the test team such as developers, business managers, customers **understand** the details of testing.

•Test Plan **guides** our thinking. It is like a rule book, which needs to be followed.

•Important aspects like test estimation, test scope, Test Strategy are **documented** in Test Plan, so it can be reviewed by Management Team and re-used for other projects.

How to write a Test Plan

You already know that making a **Test Plan** is the most important task of Test Management Process. Follow the seven steps below to create a test plan as per IEEE 829

1.Analyze the product
2.Design the Test Strategy
3.Define the Test Objectives
4.Define Test Criteria
5.Resource Planning
6.Plan Test Environment
7.Schedule & Estimation
8.Determine Test Deliverables

Test Management :

Test Management is a process where testing activities are managed to ensure high-quality and high-end testing of software applications. This method consists of tracking, organization, controlling processes and checking the visibility of the testing process to deliver a high-quality software application. It makes sure the software testing process runs as expected.

Test Management Process :

It is a software process that manages all software testing activities' from start to end. This management process provides planning, controlling, tracking, and monitoring facilities throughout the whole group cycle, these process include several activities like test case design and test execution, test planning, etc. It also gives an initial plan and discipline specifications for the software testing process.

The test management process has **two main parts of test Management Process:**

Planning :

•Risk analysis
•Test Estimation
•Test planning

Execution :

•Testing Activity
•Issue Management
•Test report and evolution

The activity of the test process :

1.Test plan –
Rough sketches serve as test plans to convey the process of testing. Gives a clear vision of the complete testing process.

2.Test design –
Test design affords the implementation process of testing.

3.Test execution –
It shows the actual system results against the expected result during test execution.

4.Exit criteria –
It gives the signal when to stop the test execution process.

5.Test reporting –
Test reporting picturizes the test process and result for the particular testing cycle.

Advantages :

- Reuses current test and compares the results with last trials.
- Prevents duplicate issues
- Enables conceptual graphical visualization regarding reports
- Reports errors via email
- Combines easily with automation tools and C
- Deals with tests according to the requirements effortlessly
- Integrated quickly with slack
- Deals test based on cycle and sprints

Disadvantages :

- It is not cost-effective
- Doesn’t support cloud-based application
- No mobile app support

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.

- It is used to automate the testing tasks that are difficult to perform manually.
- Automation tests can be run at any time of the day as they use scripted sequences to examine the software.
- Automation tests can also enter test data compare the expected result with the actual result and generate detailed test reports.
- The goal of automation tests is to reduce the number of test cases to be executed manually but not to eliminate manual testing.
- It is possible to record the test suit and replay it when required.

Automation Testing Process

1.Test Tool Selection: There will be some criteria for the Selection of the tool. The majority of the criteria include: Do we have skilled resources to allocate for automation tasks, Budget constraints, and Do the tool satisfies our needs?

2.Define Scope of Automation: This includes a few basic points such as the Framework should support Automation Scripts, Less Maintenance must be there, High Return on Investment, Not many complex Test Cases

3.Planning, Design, and Development: For this, we need to Install particular frameworks or libraries, and start designing and developing the test cases such as NUnit, JUnit, QUnit, or required Software Automation Tools

4.Test Execution: Final Execution of test cases will take place in this phase and it depends on Language to Language for .NET, we'll be using NUnit, for Java, we'll be using JUnit, for JavaScript, we'll be using QUnit or Jasmine, etc.

5.Maintenance: Creation of Reports generated after Tests and that should be documented to refer to that in the future for the next iterations.

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.
- **Saves Time and Money:** The initial investment for automation testing is on the higher side but it is cost-efficient and time-efficient in the long run. This is due to the reduction in the amount of time required for test case creation and execution which contributes to the high quality of work.
- **Earlier detection of defects:** Automation testing documents the defects, thus making it easier for the development team to fix the defect and give a faster output. The earlier the defect is identified, the more easier and cost-efficient it is to fix the defects.

Disadvantages of Automation Testing

- **High initial cost:** Automation testing in the initial phases requires a lot of time and money investment. It requires a lot of effort for selecting the tool and designing customized software.
- **100% test automation is not possible:** Generally, the effort is to automate all the test cases but in practical real situations not all test cases can be automated some test cases require human intervention for careful observation. There is always a human factor, i.e., it can't test everything like humans(design, usability, etc.).
- **Not possible to automate all testing types:** It is not possible to automate tests that verify the user-friendliness of the system. Similarly, if we talk about the graphics or sound files, even their testing cannot be automated as automated tests typically use textual descriptions to verify the output.
- **Programming knowledge is required:** Every automation testing tool uses any one of the programming languages to write test scripts. Thus, it is mandatory to have programming knowledge for automation testing.
- **False positives and negatives:** Automation tests may sometimes fail and reflect that there is some issue in the system but there is no issue present and in some cases, it may generate false negatives if tests are designed to verify that some functionality exists and not to verify that it works as expected.

Testing in Object Oriented Systems.

As information systems are becoming more complex, the object-oriented paradigm is gaining popularity because of its benefits in analysis, design, and coding. Conventional testing methods cannot be applied for testing classes because of problems involved in testing classes, abstract classes, inheritance, dynamic binding, message, passing, polymorphism, concurrency, etc.

Testing classes is a fundamentally different issue than testing functions. A function (or a procedure) has a clearly defined input-output behavior, while a class does not have an input-output behavior specification. We can test a method of a class using approaches for testing functions, but we cannot test the class using these
approaches.

- Data dependencies between variables
- Calling dependencies between modules
- Functional dependencies between a module and the variable it computes
- Definitional dependencies between a variable and its types.

But in Object-Oriented systems, there are the following additional dependencies:

- Class-to-class dependencies
- Class to method dependencies
- Class to message dependencies
- Class to variable dependencies
- Method to variable dependencies
- Method to message dependencies
- Method to method dependencies

Purpose of Object Oriented Testing

1.Object Interaction Validation: Check to make sure objects interact with one another appropriately in various situations. Testing makes ensuring that the interactions between objects in object-oriented systems result in the desired results.

2.Determining Design Errors: Find the object-oriented design's limitations and design faults. Testing ensures that the design complies with the desired architecture by assisting in the identification of problems with inheritance, polymorphism, encapsulation and other OOP concepts.

3.Finding Integration Problems: Evaluate an object's ability to integrate and communicate with other objects when it is part of a bigger component or subsystem. This helps in locating integration difficulties, such improper method calls or issues with data exchange.

4.Assessment of Reusable Code: Evaluate object-oriented code's reusability. Code reuse is promoted by object-oriented programming via features like inheritance and composition. Testing ensures that reusable parts perform as intended in various scenarios.

5.Verification of Handling Exceptions: Confirm that objects respond correctly to error circumstances and exceptions. The purpose of object-oriented testing is to make sure that the software responds carefully and is durable in the face of unforeseen occurrences or faults.

6.Verification of Uniformity: Maintain uniformity inside and between objects and the object-oriented system as a whole. Maintainability and readability are enhanced by consistency in naming standards, coding styles and compliance to design patterns.


SR
UNIVERSITY

Software Testing Methodologies

Term: 2023-24

Unit-5

Text Books: 1. Software Testing Techniques - Boris Beizer, Dreamtech, 5th edition (2020).

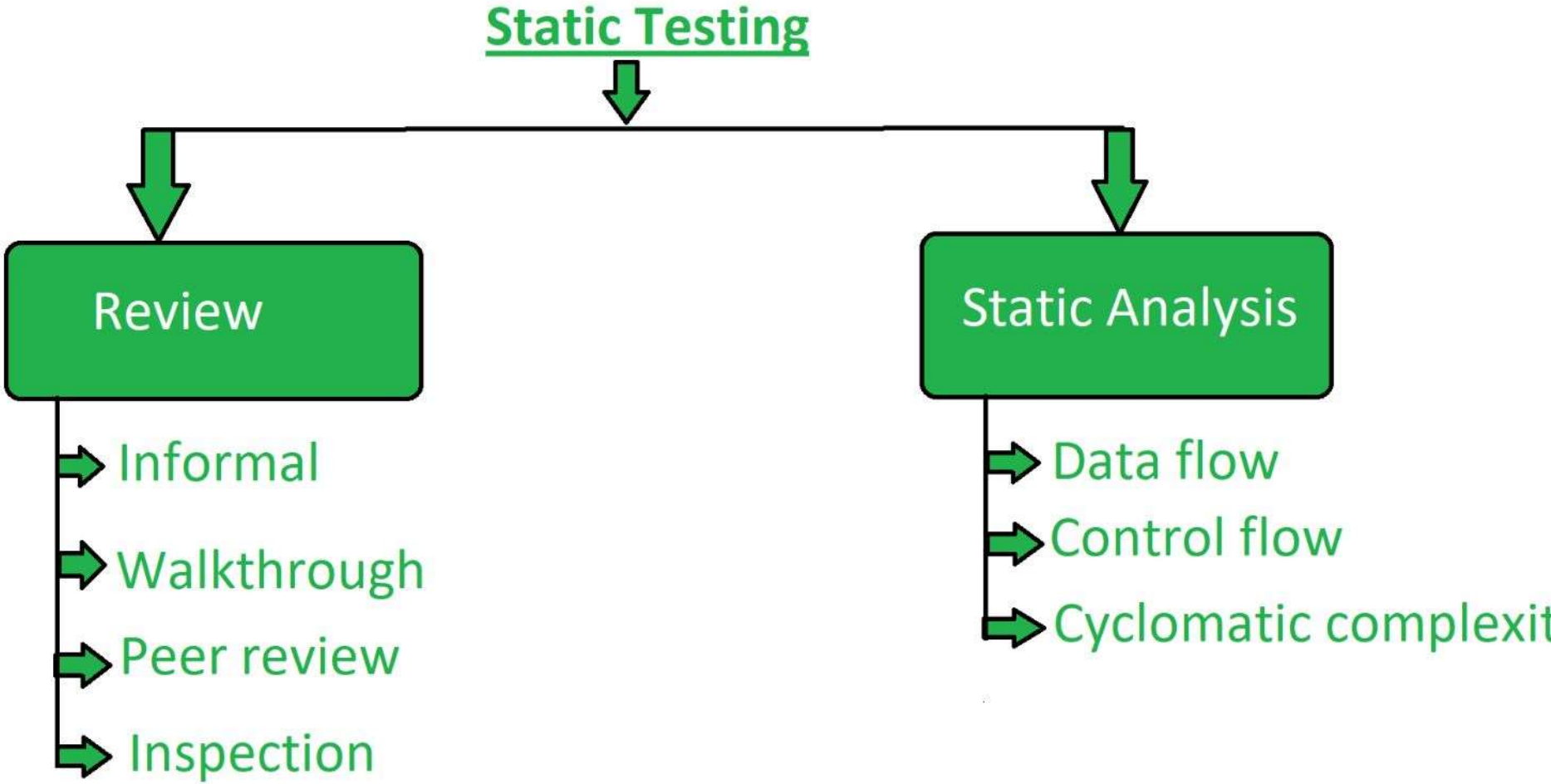
Unit-5 Syllabus

Static Testing Strategies: Formal Technical Reviews (Peer Reviews), Walk Through, Regression testing, Regression test process, Initial Smoke or Sanity test, Tools for regression testing, Ad hoc Testing: Pair testing, Exploratory testing, Iterative testing, Defect seeding. Test Planning, Management, Execution and Reporting, Software Test Automation: Testing in Object Oriented Systems.

Static Testing Strategies

- ✓ **Static Testing** is a type of a Software Testing method which is performed to check the defects in software without actually executing the code of the software application.
- ✓ Whereas in Dynamic Testing checks, the code is executed to detect the defects. Static testing is performed in early stage of development to avoid errors as it is easier to find sources of failures and it can be fixed easily.

- ✓ The errors that cannot be found using Dynamic Testing, can be easily found by Static Testing.


Static Testing
Review
Static Analysis
Informal
Walkthrough
Peer review
Inspection
Data flow
Control flow
Cyclomatic complexit

Review: In static testing review is a process or technique that is performed to find the potential defects in the design of the software. It is process to detect and remove errors and defects in the different supporting documents like software requirements specifications. People examine the documents and sorted out errors, redundancies and ambiguities. Review is of four types:

- **Informal:** In informal review the creator of the documents put the contents in front of audience and everyone gives their opinion and thus defects are identified in the early stage.

- **Walkthrough:** It is basically performed by experienced person or expert to check the defects so that there might not be problem further in the development or testing phase.

- **Peer review:** Peer review means checking documents of one-another to detect and fix the defects. It is basically done in a team of colleagues.

- **Inspection:** Inspection is basically the verification of document the higher authority like the verification of software requirement specifications (SRS).

- **Static Analysis:** Static Analysis includes the evaluation of the code quality that is written by developers. Different tools are used to do the analysis of the code and comparison of the same with the standard. It also helps in following identification of following defects:
- `(a) Unused variables (b) Dead code (c) Infinite loops (d) Variable with undefined value (e) Wrong syntax`

Static Analysis is of three types:

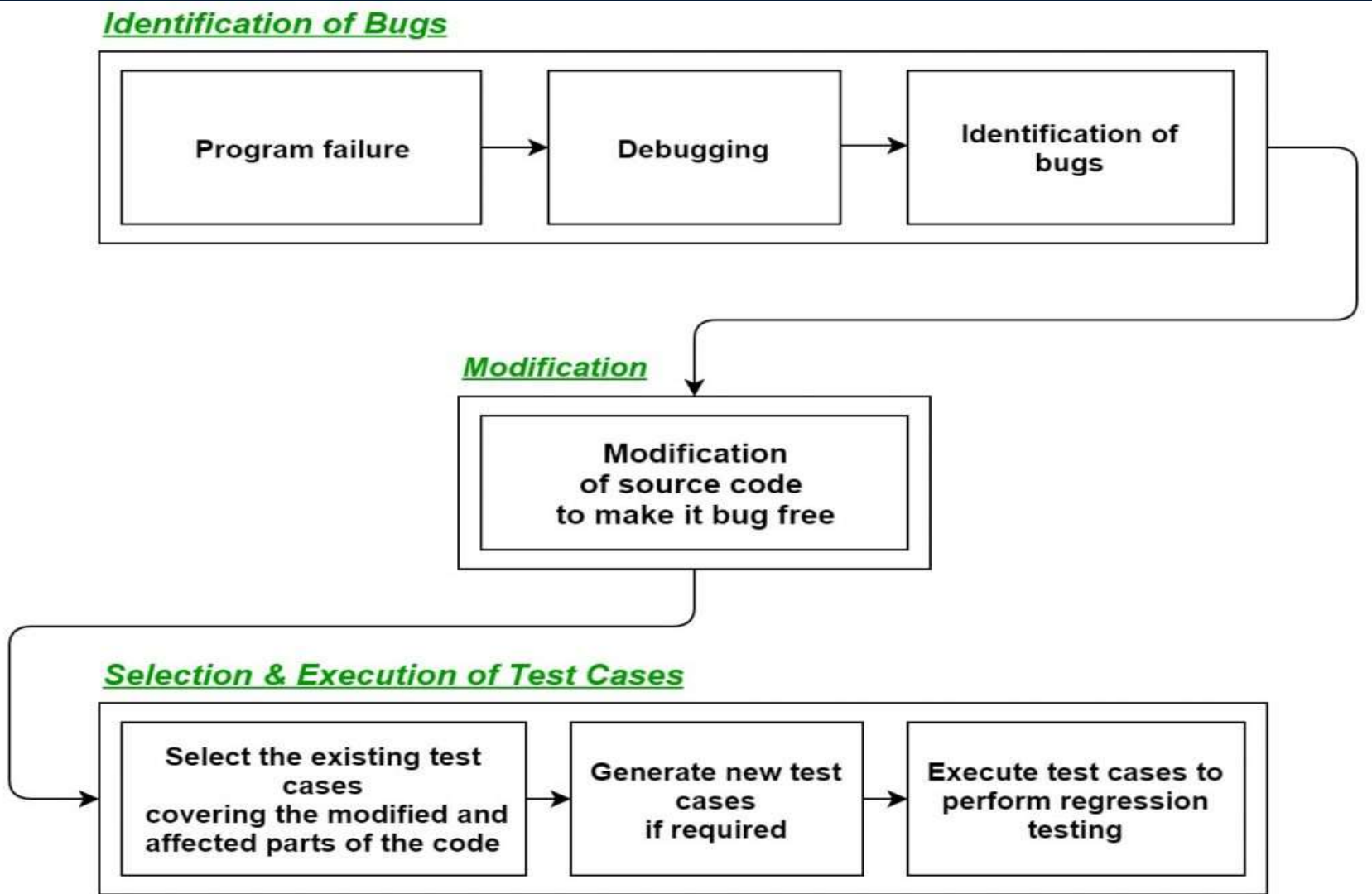
- **Data Flow:** Data flow is related to the stream processing.
- **Control Flow:** Control flow is basically how the statements or instructions are executed.
- **Cyclomatic Complexity:** Cyclomatic complexity defines the number of independent paths in the control flow graph made from the code or flowchart so that minimum number of test cases can be designed for each independent path.

Regression Testing

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

When to do regression testing?

- When new functionality is added to the system and the code has been modified to absorb and integrate that functionality with the existing code.
- When some defect has been identified in the software and the code is debugged to fix it.
- When the code is modified to optimize its working.


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

Sanity Testing and Smoke Testing

Sanity testing is a testing technique that is used to check that specific functionality or components of a software application are working as expected after making changes or fixing defects. The main objective of sanity testing is to verify that the changes made to the application have not introduced new defects or issues in the specific functionality or components. Sanity testing is typically performed after regression testing and is focused on specific areas of the application.

Key Differences between smoke and sanity

- **Objective**:
  Smoke testing aims to ensure that the build is stable enough for further testing, while sanity testing aims to verify that specific functionality or components of the application are working as expected.

- **Scope:**
  Smoke testing covers the entire system or application, whereas sanity testing is focused on specific functionality or components.
- **Timing**:
  Smoke testing is typically performed after a new build or release, while sanity testing is performed after making changes or fixing defects.
- **Depth**:
  Smoke testing is a shallow check of the software application to verify that there are no critical defects, while sanity testing is a more detailed check of specific functionality or components of the application.

In summary, smoke testing and sanity testing are both important testing techniques that are used to ensure the quality and reliability of software applications. Smoke testing is used to ensure that the build is stable enough for further testing, while sanity testing is used to verify that specific functionality or components are working as expected after making changes or fixing defects. Smoke testing is a shallow check of the entire application, while sanity testing is a more detailed check of specific functionality or components of the application.

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.

Adhoc testing has –

- No Documentation.
- No Test cases.
- No Test Design.

Adhoc testing saves a lot of time and one great example of Adhoc testing can be when the client needs the product by today 6 PM but the product development will be completed at 4 PM the same day. So in hand only limited time i.e. 2 hours only, within that 2hrs the developer and tester team can test the system as a whole by taking some random inputs and can check for any errors.

Types of Adhoc Testing

Adhoc testing is divided into three types as follows.

- **Buddy Testing** – Buddy testing is a type of Adhoc testing where two bodies will be involved one is from the Developer team and one from the tester team. So that after completing one module and after completing Unit testing the tester can test by giving random inputs and the developer can fix the issues too early based on the currently designed test cases.
- **Pair Testing** – Pair testing is a type of Adhoc testing where two bodies from the testing team can be involved to test the same module. When one tester can perform the random test another tester can maintain the record of findings. So when two testers get paired they exchange their ideas, opinions, and knowledge so good testing is performed on the module.

Monkey Testing – Monkey testing is a type of Adhoc testing in which the system is tested based on random inputs without any test cases the behavior of the system is tracked and all the functionalities of the system are working or not is monitored. As the randomness approach is followed there is no constraint on inputs so it is called Monkey testing.

Characteristics of Adhoc Testing

- Adhoc testing is performed randomly.
- Based on no documentation, no test cases, and no test designs.
- It is done after formal testing.
- It follows an unstructured way of testing.
- It takes comparatively less time than other testing techniques.
- It is good for finding bugs and inconsistencies that are mentioned in test cases.

When to conduct Adhoc testing

- When there is limited time in hand to test the system.
- When there are no clear test cases to test the product.
- When formal testing is completed.
- When the development is mostly complete.

When not to conduct Adhoc testing

- When an error exists in the test cases.
- When Beta testing is being carried out

Thank You