UNIT – 5: Software Testing and Maintenance

- Software is tested to uncover errors introduced during design and construction.
- Software testing is a process of identifying the correctness of software by considering its all attributes (Reliability, Scalability, Portability, Re-usability, Usability) and evaluating the execution of software components to find the software bugs or errors or defects.

Software testing can be divided into two steps:

1. Verification: it refers to the set of tasks that ensure that the software correctly implements a specific function.

2. Validation: it refers to a different set of tasks that ensure that the software that has been built is traceable to customer requirements.

Verification: “Are we building the product right?”
Validation: “Are we building the right product?”

Difference between Verification and Validation

Verification	Validation
Verification is the process to find whether the software meets the specified requirements for particular phase.	The validation process is checked whether the software meets requirements and expectation of the customer.
It estimates an intermediate product.	It estimates the final product.
The objectives of verification is to check whether software is constructed according to requirement and design specification.	The objectives of the validation is to check whether the specifications are correct and satisfy the business need.
It describes whether the outputs are as per the inputs or not.	It explains whether they are accepted by the user or not.
Verification is done before the validation.	It is done after the verification.
Plans, requirement, specification, code are evaluated during the verifications.	Actual product or software is tested under validation.
It manually checks the files and document.	It is a computer software or developed program based checking of files and document.

Who does Testing?

It depends on the process and the associated stakeholders of the project(s). In the IT industry, large companies have a team with responsibilities to evaluate the developed software in context of the given requirements. Moreover, developers also conduct testing which is called Unit Testing. In most cases, the following

professionals are involved in testing a system within their respective capacities −

Software Tester
Software Developer
Project Lead/Manager
End User

A STRATEGIC APPROACH TO SOFTWARE TESTING

A number of software testing strategies have been proposed in the literature.

All provide you with a template for testing and all have the following generic characteristics:

• To perform effective testing, you should conduct effective technical

reviews. By doing this, many errors will be eliminated before testing

commences.

• Testing begins at the component level and works “outward” toward the

integration of the entire computer-based system.

• Different testing techniques are appropriate for different software

engineering approaches and at different points in time.

• Testing is conducted by the developer of the software and (for large

projects) an independent test group.

• Testing and debugging are different activities, but debugging must be accommodated in any testing strategy

A strategy of software testing is shown in the context of spiral.

Following figure shows the testing strategy:

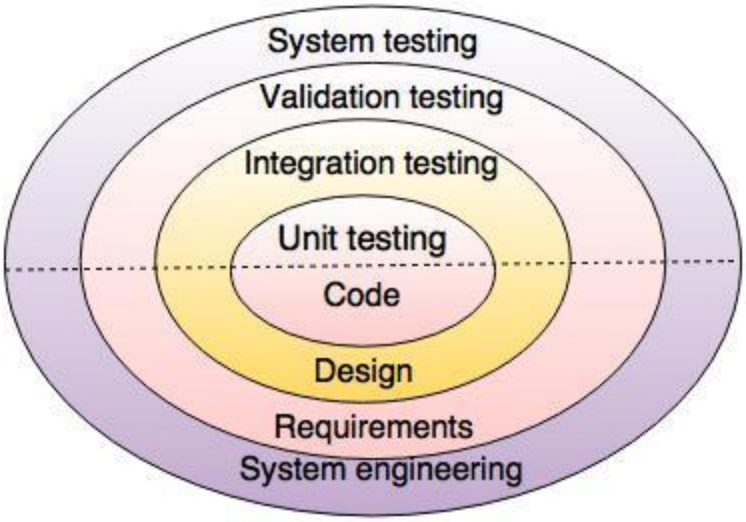


Fig. - Testing Strategy

Unit testing

Unit testing is concerned with testing the smallest modules of the software. For each module, the module's interface is examined to ensure that information properly flows to and from the module. The data structures which are internal to the module should be examined to ensure that they remain in a consistent state as the module is used.

Integration testing

Once unit testing is complete, the next important step is to combine the separate modules and ensure that they function correctly together. One may think that because

each module functions correctly by itself that they will function correctly together. This is not always the case. For example, module A might have certain expectations about the behavior of module B that are not being met, causing A to function incorrectly.

There are two general methods for performing module integration: the top-down and bottom-up approaches.

Top-down integration testing begins by creating the overall software where much of its functionality is provided by empty stub modules. These modules perform no function other than to allow the software to compile, and to provide a way to exercise the modules which are being tested. The stubs are then replaced with the actual modules, one at a time, beginning with the modules involved with user-interaction and ending with the modules which perform the software's functionality. As each module passes its tests, a new module is integrated.

Clearly, top-down integration is difficult because of the need to create stub modules. The proper testing of some modules may rely upon a wide range of behaviour from the sub-modules, and so either the testing must be delayed until the stubs have been replaced, or the stubs must offer much functionality themselves (and may themselves be buggy).

The alternative to the top-down approach is bottom-up integration testing: here, modules which do not rely on other modules are combined together to create the software. Because we begin with modules that do not rely on other modules, no stub code is needed at all. At each step we test the combined software. When all tests have passed we add another module together. Ultimately, all the modules will be combined into the functioning software.

Validation testing

Unit and integration testing asks the question, “Are we developing the software correctly?” Validation testing, on the other hand, asks, “Are we developing the correct software?” In other words, validation testing is concerned with whether the software meets its requirements.

Validation testing focuses on the user-visible portions of the software, such as the user-visible inputs and outputs, and the software's actions. The tests examine these user-visible portions to ensure that they meet the software requirements. While not part of the software itself, the documentation should also be examined to ensure that they meet any requirements concerning them.

System testing

System testing: The last high-order testing step falls outside the boundary of software engineering and into the broader context of computer system engineering. Software, once validated, must be combined with other system elements (e.g., hardware, people, databases).

System testing verifies that all elements mesh properly and that overall system function/performance is achieved.

Software Engineering-Object Oriented Testing strategies:

The classical strategy for testing computer software begins with “testing in the small” and works outward toward “testing in the large.” Stated in the jargon of software testing , we begin with unit testing, then progress toward integration testing, and culminate with validation and system testing. In conventional applications, unit testing focuses on the smallest compilable program unit—the subprogram (e.g., module, subroutine, procedure, component). Once each of these units has been tested individually, it is integrated into a program structure while a series of regression tests are run to uncover errors due to interfacing between the modules and side effects caused by the addition of new

units. Finally, the system as a whole is tested to ensure that errors in requirements are uncovered.

Unit Testing in the OO Context

When object-oriented software is considered, the concept of the unit changes. Encapsulation drives the definition of classes and objects. This means that each class and each instance of a class (object) packages attributes (data) and the operations (also known as methods or services) that manipulate these data. Rather than testing an individual module, the smallest testable unit is the encapsulated class or object. Because a class can contain a number of different operations and a particular operation may exist as part of a number of different classes, the meaning of unit testing changes dramatically.

We can no longer test a single operation in isolation (the conventional view of unit testing) but rather as part of a class. To illustrate, consider a class hierarchy in which an operation X is defined for the superclass and is inherited by a number of subclasses. Each subclass uses operation X, but it is applied within the context of the private attributes and operations that have been defined for the

subclass. Because the context in which operation X is used varies in subtle ways, it is necessary to test operation X in the context of each of the subclasses. This means that testing operation X in a vacuum (the traditional unit testing approach) is ineffective in the object-oriented context.

Class testing for OO software is the equivalent of unit testing for conventional software. Unlike unit testing of conventional software, which tends to focus on the algorithmic detail of a module and the data that flow across the module interface, class testing for OO software is driven by the operations encapsulated by the class and the state behavior of the class.

Integration Testing in the OO Context

Because object-oriented software does not have a hierarchical control structure, conventional top-down and bottom-up integration strategies have little meaning. In addition, integrating operations one at a time into a class (the conventional incremental integration

approach) is often impossible because of the “direct and indirect interactions of the components that make up the class”.

There are two different strategies for integration testing of OO systems . The first, thread-based testing, integrates the set of classes required to respond to one input or event for the system. Each thread is integrated and tested individually. Regression testing is applied to ensure that no side effects occur. The second integration approach, use-based testing, begins the construction of the system by testing those classes (called independent classes) that use very few (if any) of server classes. After the independent classes are tested, the next layer of classes, called dependent classes, that use the independent classes are tested. This sequence of testing layers of dependent lasses continues until the entire system is constructed. Unlike conventional integration, the use of drivers and stubs as replacement operations is to be avoided, when possible.

Cluster testing is one step in the integration testing of OO software. Here, a cluster of collaborating classes (determined by examining the CRC and object-relationship model) is exercised by designing test cases that attempt to uncover errors in the collaborations.

Validation Testing in an OO Context

At the validation or system level, the details of class connections disappear. Like conventional validation, the validation of OO software focuses on user-visible actions and user-recognizable output from the system. To assist in the derivation of validation tests, the tester should draw upon the use-cases that are part of the analysis model. The use-case provides a scenario that has a high likelihood of uncovered errors in user interaction requirements

There are different methods that can be used for software testing. This chapter briefly describes the methods available.

Black-Box Testing

The technique of testing without having any knowledge of the interior workings of the application is called black-box testing. The tester is oblivious to the system architecture and does not have access to the source code. Typically, while performing a black-box test, a tester will interact with the system's user interface by providing inputs and examining outputs without knowing how and where the inputs are worked upon.

The following table lists the advantages and disadvantages of black-box testing.

Advantages	Disadvantages
Well suited and efficient for large code segments.	Limited coverage, since only a selected number of test scenarios is actually performed.
Code access is not required.	

Inefficient testing, due to the fact that the tester only has limited knowledge about an application.

Clearly separates user's perspective from the developer's perspective through visibly defined roles.

Large numbers of moderately skilled testers can test the application with no knowledge of implementation, programming language, or operating systems. The test cases are difficult to design.

White-Box Testing

White-box testing is the detailed investigation of internal logic and structure of the code. White-box testing is also called glass testing or open-box testing. In order to perform white-box testing on an application, a tester needs to know the internal workings of the code.

The tester needs to have a look inside the source code and find out which unit/chunk of the code is behaving inappropriately.

The following table lists the advantages and disadvantages of white-box testing.

Advantages Disadvantages

As the tester has knowledge of the source code, it becomes very easy to find out which type of data can help in testing the application effectively. Due to the fact that a skilled tester is needed to perform white-box testing, the costs are increased.

It helps in optimizing the code. Sometimes it is impossible to look into every nook and corner to find out hidden errors that may create problems, as many paths will go untested.

Extra lines of code can be removed which can bring in hidden defects. It is difficult to maintain white-box testing, as it requires specialized tools like code analyzers and debugging tools.

Due to the tester's knowledge about the code, maximum coverage is attained during test scenario writing.

Grey-Box Testing

Grey-box testing is a technique to test the application with having a limited knowledge of the internal workings of an application. In software testing, the phrase the more you know, the better carries a lot of weight while testing an application.

Mastering the domain of a system always gives the tester an edge over someone with limited domain knowledge. Unlike black-box testing, where the tester only tests the application's user interface; in grey-box testing, the tester has access to design documents and the database. Having this knowledge, a tester can prepare better test data and test scenarios while making a test plan.

Advantages Disadvantages

Offers combined benefits of black-box and white-box testing wherever possible. Since the access to source code is not available, the ability to go over the code and test coverage is limited.

Grey box testers don't rely on the source code; instead they rely on interface definition and functional

specifications. The tests can be redundant if the software designer has already run a test case.

Based on the limited information available, a grey-box tester can design excellent test scenarios especially around communication protocols and data type handling.

Testing every possible input stream is unrealistic because it would take an unreasonable amount of time; therefore, many program paths will go untested.

The test is done from the point of view of the user and not the designer.

A Comparison of Testing Methods

The following table lists the points that differentiate black-box testing, grey-box testing, and white-box testing.

Black-Box Testing	Grey-Box Testing	White-Box Testing
The internal workings of an application need not be known.	The tester has limited knowledge of the internal workings of the application.	Tester has full knowledge of the internal workings of the application.
Also known as closed-box testing, data-driven testing, or functional testing.	Also known as translucent testing, as the tester has limited knowledge of the insides of the application.	Also known as clear-box testing, structural testing, or code-based testing.
Performed by end-users and also by testers and developers.	Performed by end-users and also by testers and developers.	Normally done by testers and developers.
Testing is based on external expectations - Internal behavior of the application is unknown.	Testing is done on the basis of high-level database diagrams and data flow diagrams.	Internal workings are fully known and the tester can design test data accordingly.

It is exhaustive and the least time-consuming.	Partly time-consuming and exhaustive.	The most exhaustive and time-consuming type of testing.
Not suited for algorithm testing.	Not suited for algorithm testing.	Suited for algorithm testing.
This can only be done by trial-and-error method.	Data domains and internal boundaries can be tested, if known.	Data domains and internal boundaries can be better tested.

The main levels of software testing are −

- Functional Testing
- Non-functional Testing

Functional Testing

This is a type of black-box testing that is based on the specifications of the software that is to be tested. The application is tested by providing input and then the results are examined that need to conform to the functionality it was intended for. Functional testing of a software is conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements.

There are five steps that are involved while testing an application for functionality.

Steps	Description
I	The determination of the functionality that the intended application is meant to perform.
II	The creation of test data based on the specifications of the application.
III	The output based on the test data and the specifications of the application.
IV	The writing of test scenarios and the execution of test cases.
V	The comparison of actual and expected results based on the executed test cases.

An effective testing practice will see the above steps applied to the testing policies of every organization and hence it will make sure that the organization maintains the strictest of standards when it comes to software quality.

Unit Testing

This type of testing is performed by developers before the setup is handed over to the testing team to formally execute the test cases. Unit testing is performed by the respective developers on the individual units of source code assigned areas. The developers use test data that is different from the test data of the quality assurance team.

The goal of unit testing is to isolate each part of the program and show that individual parts are correct in terms of requirements and functionality.

Limitations of Unit Testing

Testing cannot catch each and every bug in an application. It is impossible to evaluate every execution path in every software application. The same is the case with unit testing.

There is a limit to the number of scenarios and test data that a developer can use to verify a source code. After having exhausted all the options, there is no choice but to stop unit testing and merge the code segment with other units.

Integration Testing

Integration testing is defined as the testing of combined parts of an application to determine if they function correctly. Integration testing can be done in two ways: Bottom-up integration testing and Top-down integration testing.

Sr.No.	Integration Testing Method
1	**Bottom-up integration** This testing begins with unit testing, followed by tests of progressively higher-level combinations of units called modules or builds.
2	**Top-down integration** In this testing, the highest-level modules are tested first and progressively, lower-level modules are tested thereafter.

In a comprehensive software development environment, bottom-up testing is usually done first, followed by top-down testing. The process concludes with multiple tests of the complete application, preferably in scenarios designed to mimic actual situations.

System Testing

System testing tests the system as a whole. Once all the components are integrated, the application as a whole is tested rigorously to see that it meets the specified Quality Standards. This type of testing is performed by a specialized testing team.

System testing is important because of the following reasons −

- System testing is the first step in the Software Development Life Cycle, where the application is tested as a whole.
- The application is tested thoroughly to verify that it meets the functional and technical specifications.
- The application is tested in an environment that is very close to the production environment where the application will be deployed.
- System testing enables us to test, verify, and validate both the business requirements as well as the application architecture.

Regression Testing

Whenever a change in a software application is made, it is quite possible that other areas within the application have been affected by this change. Regression testing is performed to verify that a fixed bug hasn't resulted in another functionality or business rule violation. The intent of regression testing is to ensure that a change, such as a bug fix should not result in another fault being uncovered in the application.

Regression testing is important because of the following reasons −

- Minimize the gaps in testing when an application with changes made has to be tested.
- Testing the new changes to verify that the changes made did not affect any other area of the application.
- Mitigates risks when regression testing is performed on the application.
- Test coverage is increased without compromising timelines.
- Increase speed to market the product.

Acceptance Testing

This is arguably the most important type of testing, as it is conducted by the Quality Assurance Team who will gauge whether the application meets the intended specifications and satisfies the client’s requirement. The QA team will have a set of pre-written scenarios and test cases that will be used to test the application.

More ideas will be shared about the application and more tests can be performed on it to gauge its accuracy and the reasons why the project was initiated. Acceptance tests are not only intended to point out simple spelling mistakes, cosmetic errors, or interface gaps, but also to point out any bugs in the application that will result in system crashes or major errors in the application.

By performing acceptance tests on an application, the testing team will reduce how the application will perform in production. There are also legal and contractual requirements for acceptance of the system.

Alpha Testing

This test is the first stage of testing and will be performed amongst the teams (developer and QA teams). Unit testing, integration testing and system testing when combined together is known as alpha testing. During this phase, the following aspects will be tested in the application −

- Spelling Mistakes
- Broken Links
- Cloudy Directions
- The Application will be tested on machines with the lowest specification to test loading times and any latency problems.

Beta Testing

This test is performed after alpha testing has been successfully performed. In beta testing, a sample of the intended audience tests the application. Beta testing is also known as **pre-release testing**. Beta test versions of software are ideally distributed to a wide audience on the Web, partly to give the program a "real-world" test and partly to provide a preview of the next release. In this phase, the audience will be testing the following −

- Users will install, run the application and send their feedback to the project team.
- Typographical errors, confusing application flow, and even crashes.
- Getting the feedback, the project team can fix the problems before releasing the software to the actual users.
- The more issues you fix that solve real user problems, the higher the quality of your application will be.
- Having a higher-quality application when you release it to the general public will increase customer satisfaction.

Non-Functional Testing

This section is based upon testing an application from its non-functional attributes. Non-functional testing involves testing a software from the requirements which are nonfunctional in nature but important such as performance, security, user interface, etc.

Some of the important and commonly used non-functional testing types are discussed below.

Performance Testing

It is mostly used to identify any bottlenecks or performance issues rather than finding bugs in a software. There are different causes that contribute in lowering the performance of a software −

- Network delay
- Client-side processing
- Database transaction processing
- Load balancing between servers
- Data rendering

Performance testing is considered as one of the important and mandatory testing type in terms of the following aspects −

- Speed (i.e. Response Time, data rendering and accessing)
- Capacity
- Stability
- Scalability

Performance testing can be either qualitative or quantitative and can be divided into different sub-types such as **Load testing** and **Stress testing**.

Load Testing

It is a process of testing the behavior of a software by applying maximum load in terms of software accessing and manipulating large input data. It can be done at both normal and peak load conditions. This type of testing identifies the maximum capacity of software and its behavior at peak time.

Most of the time, load testing is performed with the help of automated tools such as Load Runner, AppLoader, IBM Rational Performance Tester, Apache JMeter, Silk Performer, Visual Studio Load Test, etc.

Virtual users (VUsers) are defined in the automated testing tool and the script is executed to verify the load testing for the software. The number of users can be increased or decreased concurrently or incrementally based upon the requirements.

Stress Testing

Stress testing includes testing the behavior of a software under abnormal conditions. For example, it may include taking away some resources or applying a load beyond the actual load limit.

The aim of stress testing is to test the software by applying the load to the system and taking over the resources used by the software to identify the breaking point. This testing can be performed by testing different scenarios such as −

- Shutdown or restart of network ports randomly

- Turning the database on or off
- Running different processes that consume resources such as CPU, memory, server, etc.

Usability Testing

Usability testing is a black-box technique and is used to identify any error(s) and improvements in the software by observing the users through their usage and operation.

According to Nielsen, usability can be defined in terms of five factors, i.e. efficiency of use, learn-ability, memory-ability, errors/safety, and satisfaction. According to him, the usability of a product will be good and the system is usable if it possesses the above factors.

Nigel Bevan and Macleod considered that usability is the quality requirement that can be measured as the outcome of interactions with a computer system. This requirement can be fulfilled and the end-user will be satisfied if the intended goals are achieved effectively with the use of proper resources.

Molich in 2000 stated that a user-friendly system should fulfill the following five goals, i.e., easy to Learn, easy to remember, efficient to use, satisfactory to use, and easy to understand.

In addition to the different definitions of usability, there are some standards and quality models and methods that define usability in the form of attributes and sub-attributes such as ISO-9126, ISO-9241-11, ISO-13407, and IEEE std.610.12, etc.

UI vs Usability Testing

UI testing involves testing the Graphical User Interface of the Software. UI testing ensures that the GUI functions according to the requirements and tested in terms of color, alignment, size, and other properties.

On the other hand, usability testing ensures a good and user-friendly GUI that can be easily handled. UI testing can be considered as a sub-part of usability testing.

Security Testing

Security testing involves testing a software in order to identify any flaws and gaps from security and vulnerability point of view. Listed below are the main aspects that security testing should ensure −

- Confidentiality
- Integrity
- Authentication
- Availability
- Authorization
- Non-repudiation
- Software is secure against known and unknown vulnerabilities

- Software data is secure
- Software is according to all security regulations
- Input checking and validation
- SQL insertion attacks
- Injection flaws
- Session management issues
- Cross-site scripting attacks
- Buffer overflows vulnerabilities
- Directory traversal attacks

Portability Testing

Portability testing includes testing a software with the aim to ensure its reusability and that it can be moved from another software as well. Following are the strategies that can be used for portability testing −

- Transferring an installed software from one computer to another.
- Building executable (.exe) to run the software on different platforms.

Portability testing can be considered as one of the sub-parts of system testing, as this testing type includes overall testing of a software with respect to its usage over different environments. Computer hardware, operating systems, and browsers are the major focus of portability testing. Some of the pre-conditions for portability testing are as follows −

- Software should be designed and coded, keeping in mind the portability requirements.
- Unit testing has been performed on the associated components.
- Integration testing has been performed.
- Test environment has been established.

What is Web Application Testing?

Web application testing, a software testing technique exclusively adopted to test the applications that are hosted on web in which the application interfaces and other functionalities are tested.

Web Application Testing - Techniques:

1. **Functionality Testing** - The below are some of the checks that are performed but not limited to the below list:

- Verify there is no dead page or invalid redirects.
- First check all the validations on each field.
- Wrong inputs to perform negative testing.

- Verify the workflow of the system.
- Verify the data integrity.

2. **Usability testing -** To verify how the application is easy to use with.

- Test the navigation and controls.
- Content checking.
- Check for user intuition.

3. **Interface testing -** Performed to verify the interface and the dataflow from one system to other.

4. **Compatibility testing-** Compatibility testing is performed based on the context of the application.

- Browser compatibility
- Operating system compatibility
- Compatible to various devices like notebook, mobile, etc.

5. **Performance testing -** Performed to verify the server response time and throughput under various load conditions.

- **Load testing -** It is the simplest form of testing conducted to understand the behaviour of the system under a specific load. Load testing will result in measuring important business critical transactions and load on the database, application server, etc. are also monitored.
- **Stress testing -** It is performed to find the upper limit capacity of the system and also to determine how the system performs if the current load goes well above the expected maximum.
- **Soak testing -** Soak Testing also known as endurance testing, is performed to determine the system parameters under continuous expected load. During soak tests the parameters such as memory utilization is monitored to detect memory leaks or other performance issues. The main aim is to discover the system's performance under sustained use.
- **Spike testing -** Spike testing is performed by increasing the number of users suddenly by a very large amount and measuring the performance of the system. The main aim is to determine whether the system will be able to sustain the work load.

6. **Security testing -** Performed to verify if the application is secured on web as data theft and unauthorized access are more common issues

Debugging

Debugging is the process of identifying and resolving errors, or bugs, in a software system. It is an important aspect of software engineering because bugs can cause a software system to malfunction, and can lead to poor performance or incorrect results. Debugging can be a time-consuming and complex task, but it is essential for ensuring that a software system is functioning correctly.

There are several common methods and techniques used in debugging, including:

Code Inspection: This involves manually reviewing the source code of a software system to identify potential bugs or errors.

Debugging Tools: There are various tools available for debugging such as debuggers, trace tools, and profilers that can be used to identify and resolve bugs.

Unit Testing: This involves testing individual units or components of a software system to identify bugs or errors.

Integration Testing: This involves testing the interactions between different components of a software system to identify bugs or errors.

System Testing: This involves testing the entire software system to identify bugs or errors.

Monitoring: This involves monitoring a software system for unusual behavior or performance issues that can indicate the presence of bugs or errors.

Logging: This involves recording events and messages related to the software system, which can be used to identify bugs or errors.

Debugging Process: Steps involved in debugging are:

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

The debugging process will always have one of two outcomes :

1. The cause will be found and corrected.

2. The cause will not be found.

Debugging Approaches/Strategies:

Brute Force: Study the system for a larger duration in order to understand the system. It helps the debugger to construct different representations of systems to be debugging depending on the need. A study of the system is also done actively to find recent changes made to the software.

Backtracking: Backward analysis of the problem which involves tracing the program backward from the location of the failure message in order to identify the region of faulty code. A detailed study of the region is conducted to find the cause of defects.

Forward analysis of the program involves tracing the program forwards using breakpoints or print statements at different points in the program and studying the results. The region where the wrong outputs are obtained is the region that needs to be focused on to find the defect.

Using the past experience of the software debug the software with similar problems in nature. The success of this approach depends on the expertise of the debugger.

Cause elimination: it introduces the concept of binary partitioning. Data related to the error occurrence are organized to isolate potential causes.

Software Maintenance

Software maintenance is a part of the Software Development Life Cycle. Its primary goal is to modify and update software application after delivery to correct errors and to improve performance. Software is a model of the real world. When the real world changes, the software require alteration wherever possible.

Software Maintenance is an inclusive activity that includes error corrections, enhancement of capabilities, deletion of obsolete capabilities, and optimization.

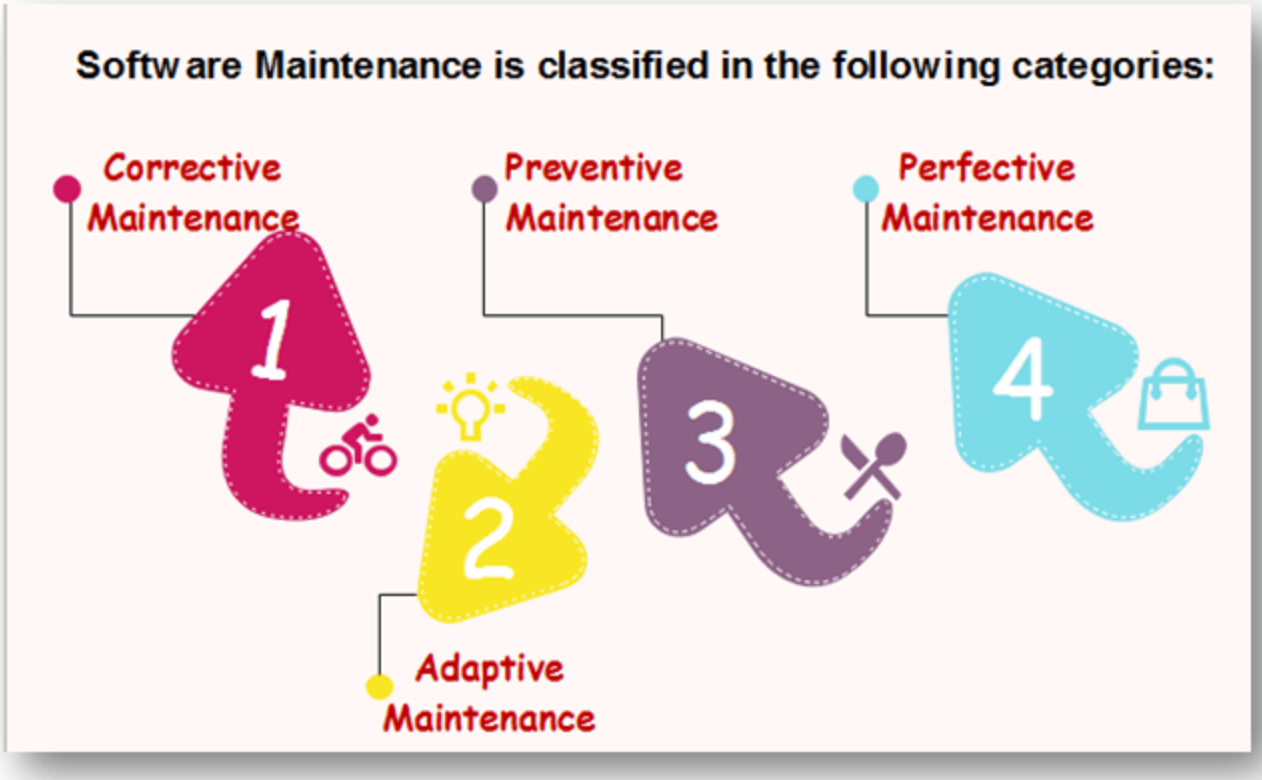
Need for Maintenance

Software Maintenance is needed for:-

- Correct errors
- Change in user requirement with time
- Changing hardware/software requirements
- To improve system efficiency
- To optimize the code to run faster
- To modify the components
- To reduce any unwanted side effects.

Thus the maintenance is required to ensure that the system continues to satisfy user requirements.

Types of Software Maintenance



1. Corrective Maintenance

Corrective maintenance aims to correct any remaining errors regardless of where they may cause specifications, design, coding, testing, and documentation, etc.

2. Adaptive Maintenance

It contains modifying the software to match changes in the ever-changing environment.

3. Preventive Maintenance

It is the process by which we prevent our system from being obsolete. It involves the concept of reengineering & reverse engineering in which an old system with old technology is re-engineered using new technology. This maintenance prevents the system from dying out.

4. Perfective Maintenance

It defines improving processing efficiency or performance or restricting the software to enhance changeability. This may contain enhancement of existing system functionality, improvement in computational efficiency, etc.

Software Engineering | Re-engineering

Software Re-engineering is a process of software development which is done to improve the maintainability of a software system. Re-engineering is the examination and alteration of a system to reconstitute it in a new form. This process encompasses a combination of sub-processes like reverse engineering, forward engineering, reconstructing etc. Re-engineering, also known as reverse engineering or software re-engineering, is the process of analyzing, designing, and modifying existing software systems to improve their quality, performance, and maintainability. This can include updating the software to work with new hardware or software platforms, adding new features, or improving the software's overall design and architecture.

Re-engineering can be done for a variety of reasons, such as:

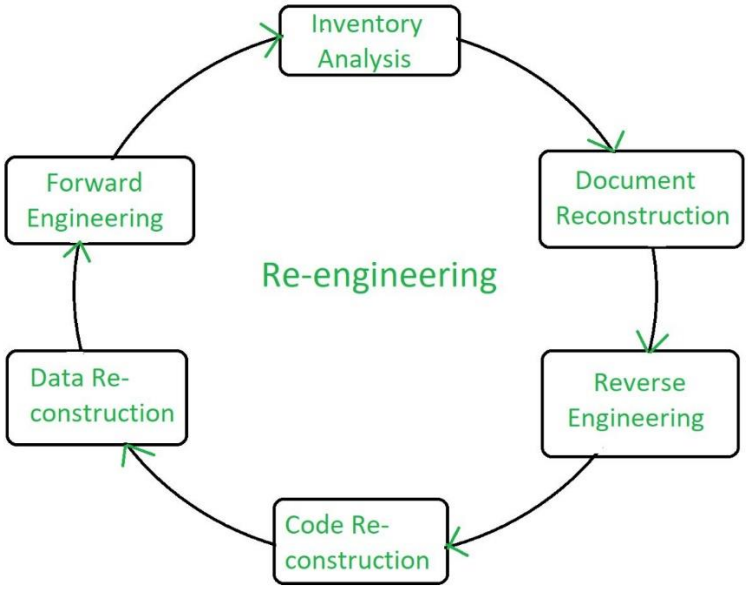
To improve the software's performance and scalability: By analyzing the existing code and identifying bottlenecks, re-engineering can be used to improve the software's performance and scalability.

1. **To add new features:** Re-engineering can be used to add new features or functionality to existing software.
2. **To support new platforms**: Re-engineering can be used to update existing software to work with new hardware or software platforms.
3. **To improve maintainability:** Re-engineering can be used to improve the software's overall design and architecture, making it easier to maintain and update over time.
4. **To meet new regulations and compliance**: Re-engineering can be done to ensure that the software is compliant with new regulations and standards.
5. Re-engineering can be a complex and time-consuming process, and it requires a thorough understanding of the existing software system. It also requires a structured and disciplined approach to software development, similar to software engineering.
6. Re-engineering can be beneficial in many cases, it can help to improve the quality, performance, and maintainability of existing software systems, but it also has its own drawbacks, such as:
7. **High costs:** Re-engineering can be a resource-intensive process and can require a significant investment in tools and training.
8. Risk of introducing new bugs: Changing existing code can introduce new bugs and compatibility issues.

Steps involved in Re-engineering:

1. Inventory Analysis
2. Document Reconstruction
3. Reverse Engineering
4. Code Reconstruction
5. Data Reconstruction
6. Forward Engineering

Diagrammatic Representation:



1. Inventory analysis:

1. An inventory should be made to store all the applications in every organization.
2. The inventory can be a spreadsheet model containing information that provides a detailed description of every active application.
3. By sorting this information according to business criticality, longevity, current maintainability, and other important criteria, candidates for re-engineering appear.
4. Resources can be allocate to candidate applications for re-engineering work.

2. Document restructuring:

1. Documentation of a system helps in explaining either how it operates or how to use it.
2. Documentation must be update.
3. Full re-documentation of an application may not be necessary. Rather, those portions of the system that are currently undergoing change are fully document.
4. Over time, a collection of useful and relevant documentation will evolve.

3. Reverse engineering:

1. Reverse engineering is a process of design recovery.
2. The reverse engineering tools extract data, architectural, and procedural design information from an existing program.

4. Code restructuring:

1. To accomplish code restructuring, the source code is analyze using a restructuring tool. Violations of structured programming constructs are noted, and code is then restructured.
2. The resultant restructured code is reviewed and tested to ensure that no anomalies have been introduced. Internal code documentation is updated.

5. Data restructuring:

1. Data restructuring begins with a reverse engineering activity.
2. Current data architecture is dissected, and necessary data models are defined.
3. Data objects and attributes are identifies, and existing data structures are reviews for quality.

6. Forward engineering:

1. Forward engineering, also called renovation or reclamation recovers design information from existing software.
2. It uses this information to alter or reconstitute the existing system in an effort to improve its overall quality.