Quote of the day

Chapter 4 – Defect Management

What is defect ?

A **Defect in Software Testing** is a variation or deviation of the software application from end user's requirements or original business requirements. A software defect is an error in coding which causes incorrect or unexpected results from a software program which does not meet actual requirements.

Why Does Software Have Bugs?

There are many reasons for the occurrence of Software Bugs. The most common reason is human mistakes in software design and coding.

Once you get to know the causes for Software Defects, then it will be easier for you to take corrective actions to minimize these defects.

#1) Miscommunication or No Communication

The success of any software application depends on the communication between stakeholders, development, and testing teams. Unclear requirements and misinterpretation of requirements are the two major factors that causedefects in software.

Also, defects are introduced in the development stage if the exact requirements are not communicated properly to the development teams.

#2) Software Complexity

The complexity of the current software applications can be difficult for anyone with no experience in modern-day software development.

Windows-type interfaces, Client-Server, and Distributed Applications, Data Communications, enormous relational databases, and sheer size of applications have all contributed to the exponential growth in software/system complexity..

#3) Programming Errors

Programmers, like anyone else, can make common programming mistakes. Not all developers are domain experts. Inexperienced programmers or programmers without proper domain knowledge can introduce simple mistakes while coding.

Lack of simple coding practices, unit testing, debugging are some of the common reasons for these issues to get introduced at the development stage.

#4) Changing Requirements

The customer may not understand the effects of changes or may understand and anyway request them to redesign, rescheduling of engineers, effects on the other projects, and the work already completed may have to be redone or thrown out, hardware requirements that may be affected, etc

#5) Time Pressures

Scheduling software projects is difficult, often requiring a lot of guesswork. When deadlines loom and the crunch comes, mistakes will be made still.

Unrealistic schedules, though not common, the major concern in small-scale projects/companies results in software bugs. If there is not enough time for proper design, coding, and testing, then it's quite obvious for defects to be introduced.

#6) Egotistical or Overconfident People

People prefer to say things like:

- 'no problem'
- 'piece of cake'
- 'I can whip that out in a few hours'
- 'It should be easy to update that old code'

Instead of:

- 'That adds a lot of complexity and we could end up making a lot of mistakes'.
- 'We do not know if we can do that; we'll wing it'.
- 'I can't estimate how long it will take until I take a closer look at it'.
- 'We can't figure out what that old spaghetti code did in the first place'.
- If there are too many unrealistic 'no problem's', then it results in softwarebugs.

#7) Poorly Documented C

It's tough to maintain and modify the code that is badly written or poorly documented; the result is ***Software Bugs***. In many organizations, management provides no incentive for programmers to document their code or write clear, understandable code.

In fact, it's usually the opposite: they get points mostly for quickly turning out code, and there's job security if nobody else can understand it ('if it was hard to write, it should be hard to read').

#8) Software Development Tools

Visual tools, class libraries, compilers, scripting tools, etc. often introduce their own bugs or are poorly documented, resulting in added bugs.

Continuously changing software tools are used by software programmers. Keeping pace with the different versions and their compatibility is a major ongoing issue.

#9) Obsolete Automation Scripts

Writing automation scripts takes a lot of time, especially for complex scenarios. If automation team's record/write any test script but forget to update it over a period, then that test could become obsolete.

Types of Defect

Following are some of the basic types of defects in the software development:

1. Arithmetic Defects:

It include the defects made by the developer in some arithmetic expression or mistake in finding solution of such arithmetic expression. This type of defects are basically made by the programmer due to access work or less knowledge. Code congestion may also lead to the arithmetic defects as programmer is unable to properly watch the written code.

2. Logical Defects:

Logical defects are mistakes done regarding the implementation of thecode. When the programmer doesn't understand the problem clearly or thinks in a wrong way then such types of defects happen. Also while implementing the code if the programmer doesn't take care of the corner cases then logical defects happen. It is basically related to the core of the software.

3. Syntax Defects:

Syntax defects means mistake in the writing style of the code. It also focuses on the small mistake made by developer while writing the code. Often the developers do the syntax defects as there might be some small symbols escaped. For example, while writing a code in C++ there is possibility that a semicolon(;) is escaped.

4. Multithreading Defects:

Multithreading means running or executing the multiple tasks at the same time. Hence in multithreading process there is possibility of the complex debugging. In multithreading processes sometimes there is condition of the deadlock and the starvation is created that may lead to system's failure.

5. Interface Defects:

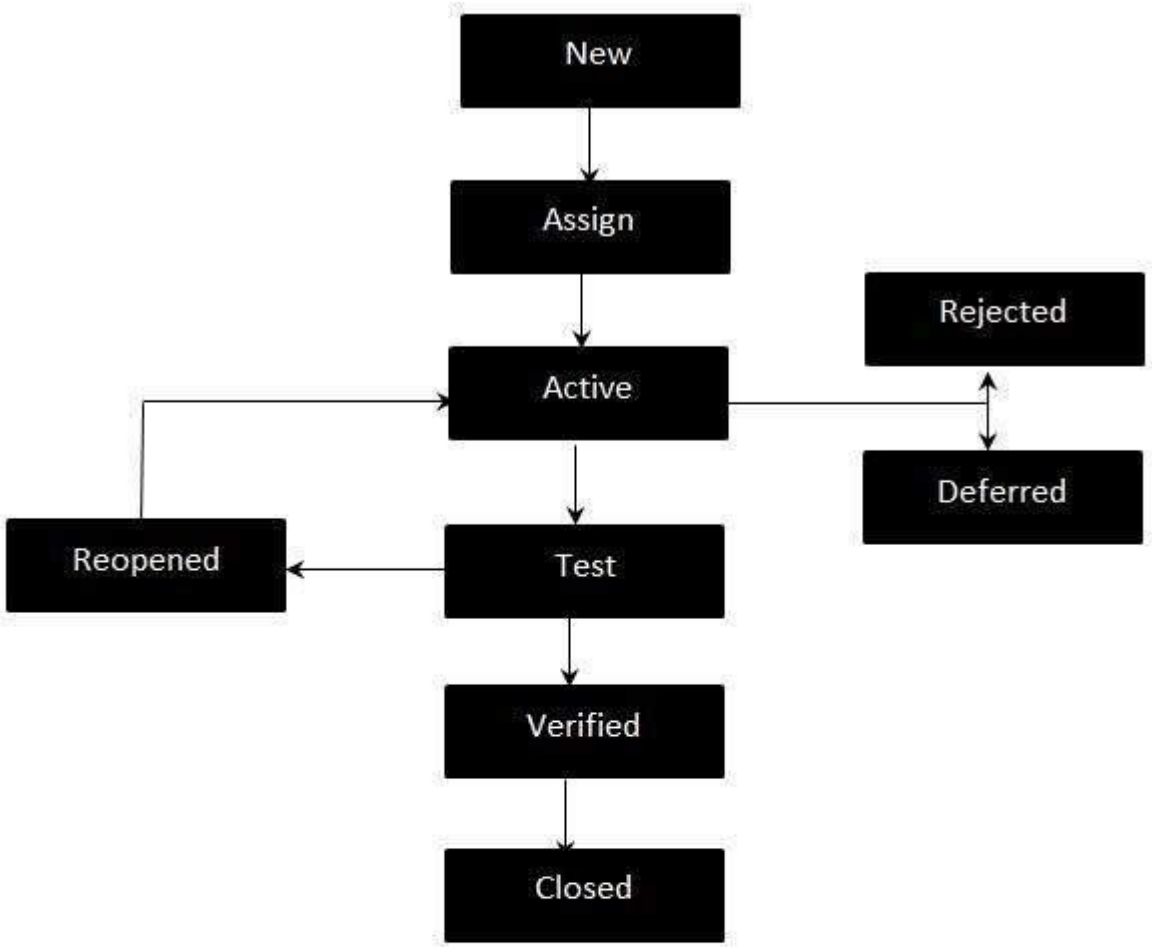
Interface defects means the defects in the interaction of the software and the users. The system may suffer different kinds of the interface testing in the forms of the complicated interface, unclear interface or the platform based interface.

6. Performance Defects:

Performance defects are the defects when the system or the software application is unable to meet the desired and the expected results. When the system or the software application doesn't fulfill the users's requirements then that is the performance defects. It also includes the response of the system with the varying load on the system.

Defect Life Cycle - Workflow:

Defect life cycle, also known as Bug Life cycle is the journey of a defect cycle, which a defect goes through during its lifetime. It varies from organization to organization and also from project to project as it is governed by the software testing process and also depends upon the tools used.



Defect Life Cycle States:

- **New -** Potential defect that is raised and yet to be validated.
- **Assigned -** Assigned against a development team to address it but not yet resolved.
- **Active -** The Defect is being addressed by the developer and investigation is under progress. At this stage there are two possible outcomes; viz - Deferred or Rejected.
- **Test -** The Defect is fixed and ready for testing.
- **Verified -** The Defect that is retested and the test has been verified by QA.
- **Closed -** The final state of the defect that can be closed after the QA retesting or can be closed if the defect is duplicate or considered as NOT a defect.
- **Reopened -** When the defect is NOT fixed, QA reopens/reactivates the defect.
- **Deferred -** When a defect cannot be addressed in that particular cycle it is deferred to future release.
- **Rejected -** A defect can be rejected for any of the 3 reasons; viz - duplicate defect, NOT a Defect, Non Reproducible.

Defect Management
Process

The **Defect Management Process** is process where most of the organizations manage the **Defect Discovery, Defect Removal**, and then the **Process Improvement**.

As the name recommends**, the Defect Management Process (DMP)** manages defects by purely detecting and resolving or fixing the faults. It is impossible to make a software 100% error or defect-free, but several defects can be declined by fixing or resolving them.

The defect management process primarily focuses on stopping defects, finding defects in the earlier stages, and moderating the effect of defects.

The Objective of Defect Management Process (DMP)

The main objective of the defect management process is as discussed below:

- The primary objective of DMP is to expose the defects at an early stage of the software development process.
- The execution of the defect management process will help us enhance the process and implementation of software.
- The defect management process reduces the impact or effects of defects on software.
- The Defect management process (DMP) helps us to avoid defects.
- The main goal of the Defect management process is to resolve or fixing the defects.

Various Stages of Defect Management Process

The defect management process includes several stages, which are as follows:

1. Defect Prevention
2. Deliverable Baseline
3. Defect Discovery
4. Defect Resolution
5. Process Improvement
6. Management Reporting

1. Defect Prevention

The first stage of the **defect management process** is **defect prevention**. In this stage, the execution of procedures, methodology, and standard approaches decreases the risk of defects. Defect removal at the initial phase is the best approach in order to reduction its impact.

The defect prevention stage includes the following significant steps:

- **Estimate Predictable Impact**
- **Minimize expected impact**
- **Identify Critical Risk**

2. Deliverable Baseline

The second stage of the defect management process is the **Deliverable baseline.**

Here, the deliverable defines the **system, documents, or product**.

We can say that the **deliverable is a baseline** as soon as a deliverable reaches its pre-defined milestone.In this stage, the deliverable is carried from one step to another; the system's existing defects also move forward to the next step or a milestone.

3. Defect Discovery

The next stage of the defect management process is **defect discovery**. At the early stage of the defect management process, defect discovery is very significant. And later, it might cause greater damage.

The following phases have been including in the defect discovery stage; let's understand them in details:

- **Identify a defect**
- **Report a defect**
- **Acknowledge Defect**

4. Defect Resolution

Once the **defect discovery** stage has been completed successfully, we move to the next step of the defect management process, **Defect Resolution**.
The **Defect Resolution** is a step-by-step procedure of fixing the defects, or we can say that this process is beneficial in order to specified and track the defects. This process begins with handing over the defects to the development team. The developers need to proceed with the resolution of the defect and fixed them based on the **priority.**
We need to follow the below steps in order to accomplish the defect resolution stage.

- **Prioritize the risk**
- **Fix the defect**
- **Report the Resolution**

5. Process Improvement

In the above stage (defect resolution), the defects have been arranged and fixed.

Now, in the **process improvement** phase, we will look into the lower priority defects because these defects are also essential as well as impact the system.

All the acknowledged defects are equal to a critical defect from the process improvement phase perspective and need to be fixed.
The people involved in this particular stage need to recall and check from where the defect was initiated.

6. Management Reporting

Management reporting is the last stage of the **defect management process**. It is a significant and essential part of the defect management process. The management reporting is required to make sure that the generated reports have an objective and increase the defect management process.

Defect Reporting

- **DefectId** – A unique identifier of the defect.
- **Summary** – A one-line summary of the defect, more like a defect title.
- **Description** – A detailed description of the defect.
- **Build Version** – Version of the build or release in which defect is found.
- **Steps to reproduce** – The steps to reproduce the defect.
- **Expected Behavior** – The expected behavior from which the applicationis deviating because of the defect.
- **Actual Behavior** – The current erroneous state of the application w.r.t.the defect.
- **Priority** – Based on the urgency of the defect, this field can be set on ascale of P0 to P3.
- **Severity** – Based on the criticality of the defect, this field can be set to minor, medium, major or show stopper.
- **Reported By** – Name of the QA, reporting the defect.
- **Reported On** – The date on which the defect was raised.
- **Assigned To** – The person to whom the defect is assigned in its current state. It can be the developer fixing the defect, the QA for verification of the fixed defect or the manager approving the defect.
- **Current Status** – The current status of the defect (one of the states of the defect life cycle).
- **Environment** – The environment in which the defect is found – release, staging, production, etc.

Techniques to Identify

Different techniques are used to identify defect. These techniques are categorized into three categories as given below -:

1. Static Techniques :

Static technique, as name suggests, is a technique in which software is tested without any execution or program or system. In this, software products are tested or examined manually, or with help of different automation tools that are available, but it's not executed.

Different type of causes of defects can be found by this technique such as -:

- Missing requirements
- Design defects
- Deviation from standards
- Inconsistent interface specification
- Non-maintainable code
- Insufficient maintainability, etc.

2. Dynamic Techniques :

Dynamic technique, as name suggests, is a technique in which software is tested by execution of program or system. This technique can only be applied to software code as in this technique, testing is done only by execution of program or software code. Different types of defects can be found by this technique such as :

- **Functional defects –**

These defects arise when functionality of system or software does not work as per Software Requirement Specification (SRS). Defects that are very critical largely affect essential functionalities of the system that in turn affect a software product or its functionality. Software product might not work properly and stop working. These defects are simply related to working of system.

- **Non-functional defects -**

A defect in software products largely affects its non-functional aspects. These defects can affect performance, usability, etc.

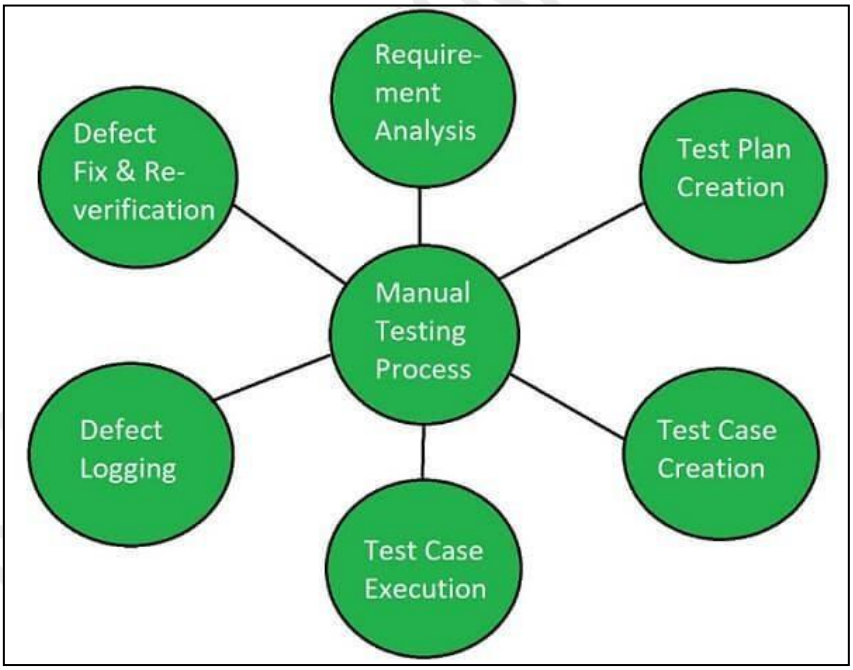
3. Operational Techniques :

Operational techniques, as name suggests, are a technique that produces a deliverable i.e. Product. Then user, customer, or control personnel identify or found defects by inspection, checking, reviewing, etc. In

5. Testing tools & Measurements

Manual Testing

Manual testing is a software testing method in which testers, often referred to asquality assurance (QA) engineers or testers, manually execute test cases withoutthe use of automation tools or scripts. The primary goal of manual testing is to identify defects, bugs, or issues in a software application by simulating user interactions and evaluating the software's behavior.



It involves the following key activities:

1. Test Case Design: Testers create detailed test cases, test scenarios, and testscripts based on software requirements and design specifications. These test cases outline the steps to be followed and the expected outcomes.

2. Test Execution: Testers execute the test cases by interacting with the software as end-users would. This involves various operations such as inputtingdata, clicking buttons, navigating menus, and performing other functions to validate the software's functionality.

3. Defect Identification: During test execution, testers actively look for and document defects or unexpected behavior in the software. They record information about the defects, including steps to reproduce the issue, severity,and other relevant details.

4. Regression Testing: Testers may perform regression testing to ensure that therecent changes or bug fixes in the software do not introduce new defects or affect existing functionality negatively.

5. Exploratory Testing: In addition to predefined test cases, testers may also conduct exploratory testing to uncover defects that might not be covered by thescripted tests. This approach allows for more flexible and creative testing.

6. Usability Testing: Testers assess the software's user-friendliness, user interface design, and overall user experience to ensure it meets the intended userneeds.

Advantages of Manual Testing

Manual testing offers several advantages that make it an indispensable part of the software testing process. Here are some of the key advantages of manual testing:

1. **Human Intelligence:** Manual testing requires human intelligence to developtest cases, identify faults, and judge the overall product quality. In order totest the program's usability, performance, and functionality, testers might make use of their domain knowledge, prior experience, and ingenuity.
2. **Adaptability:** Manual testing is adaptable and can be utilized in a range of testing scenarios. Testers are able to modify test cases, as opposed to automated testing, which cannot be modified to meet changing software needs. Additionally, exploratory testing and ad-hoc testing, which are useful for identifying fresh problems and validating the user experience, can be done manually.
3. **Cost-Effective:** Manual testing is cost-effective, especially for small-scale projects or when there is no clear understanding of the system requirements. Manual testing does not require any investment in expensive tools or automation frameworks, which can significantly reduce testing costs.
4. **Personal Touch:** Manual testing gives the software's quality a personal touch. Automated testing is unable to capture tester comments on the software's usability, aesthetics, and user experience. This comments will help us enhance the software's

functionality and guarantee a better user experience.

5. **Early Detection:** Using manual testing, serious flaws in software can be found quickly. Human testers undertake manual testing, which allows them to find flaws that an automated test could have missed. By preventing expensive rework later in the development cycle, early defect discovery can reduce development cycle time and expense.
6. **Simplicity:** Manual testing is uncomplicated and simple to comprehend. Anyone with a fundamental understanding of the software can perform manual testing; it does not call for any particular knowledge or training. Because of its ease of use, In conclusion, manual testing has a number of benefits that make it a crucial step in the software testing process.

Disadvantages of manual testing

1. **Time-consuming:** Manual testing requires time since test cases must be executed manually. Complex software program testing could take some time. Testing teams might not have enough time to cover all test cases because of the delay in software development.
2. **Human Error:** Human errors can happen when testing is done manually. By failing to test particular scenarios or by making mistakes when executing test cases, testers may come up with erroneous results. These mistakes can make it impossible to find flaws, which would affect the caliber of the software.
3. **Expensive:** Manual testing can be expensive, particularly for big projects or when frequent releases are required. It can raise testing costs because it calls for a sizable amount of resources, such as employees, time, and equipment.
4. **Difficult to Measure:** It is challenging to quantify the manual testing process since there are no objective metrics provided by manual testing to assess the software's quality. It is difficult to assess the efficiency of the testing process since it is difficult to keep track of the quantity of test cases executed, errors discovered, and test coverage attained.

Automation Testing

Automation testing is a software testing technique in which automated testing tools and scripts are used to perform test cases and verify the functionality of a software application. Instead of manual testing, where human testers interact with the software's user interface to validate its behavior, automation testing relies on pre-written scripts and test frameworks to execute test cases.

The primary objectives of automation testing are to:

1. Increase Efficiency: Automation testing can quickly and repeatedly execute a large number of test cases, making it suitable for regression testing, performance testing, and load testing, where efficiency is crucial.

2. Enhance Reproducibility: Automated tests provide consistent and reproducible results since they follow predefined steps and conditions, reducing the likelihood of human error.

3. Improve Test Coverage: Automation allows for comprehensive test coverage by running a wide range of test cases, including those that are time- consuming or complex to execute manually.

4. Accelerate Testing Cycles: Automation speeds up the testing process, allowing for faster feedback and reduced time-to-market for software products.

5. Continuous Integration and Continuous Delivery (CI/CD): Automation is integral to CI/CD pipelines, enabling frequent and automated testing to ensure that new code changes do not introduce regressions.

6. Load and Performance Testing: Automation tools can simulate a large number of users and test scenarios to assess the performance, scalability, and reliability of the software.

7. Data-Driven Testing: Automation testing supports data-driven testing, where test data is separated from test scripts, making it easy to execute the same tests with various data sets.

8. Parallel Testing: Automated tests can run in parallel on multiple environments, configurations, or devices, improving testing efficiency

Advantages of Automation Testing

1. **Efficiency:** Automation testing significantly speeds up the testing process, allowing for the execution of a large number of test cases in a short amount of time. This is particularly beneficial for regression testing, where existing functionality is validated after code changes.

2. **Reproducibility:** Automated tests follow predefined scripts and conditions,

3. **Increased Test Coverage:** Automation enables the execution of a wide range of test cases, including those that are complex, time-consuming, or repetitive.

4. **Faster Feedback:** Automation provides quicker feedback on the quality of the software, enabling faster bug detection and resolution. This is especially important in the context of continuous integration and continuous delivery (CI/CD).

5. **Cost Savings:** Over the long term, automation can lead to cost savings as it reduces the reliance on manual testers and allows for more efficient use of testing resources.

6. **Parallel Testing:** Automation tools can run tests in parallel on multiple environments or configurations, further improving efficiency.

7. **Data-Driven Testing:** Automation supports data-driven testing, making it easy to test the same functionality with various sets of test data.

8. **Performance and Load Testing:** Automation is crucial for performance and load testing, as it can simulate a large number of users and scenarios, helping identify performance bottlenecks.

Disadvantages of Automation Testing

1. **Initial Setup Time:** Creating and maintaining automated test scripts can be time-consuming and may require significant upfront effort. This can delay the start of testing, especially for projects with tight deadlines.

2. **Script Maintenance:** Automated tests need regular maintenance to adapt to changes in the application, such as updates, enhancements, or bug fixes. Maintenance efforts can become a significant overhead.

3. **Lack of Human Judgment:** Automation cannot replace human testers' ability to apply domain knowledge, intuition, and creativity to uncover unexpected defects or usability issues.

4. **Cost of Tools and Training:** Acquiring and maintaining automation tools can be costly. Training testers and developers to use these tools effectively may also require

Static & Dynamic Testing Tools

Static testing tools and dynamic testing tools are two distinct categories of software testing tools used to assess the quality and behavior of software applications. They serve different purposes in the software development and testing process.

Static Testing Tools

Static testing tools are used to analyze the source code, design documents, and other artifacts of a software application without executing the code. These tools are primarily focused on identifying issues in the code, design, or documentation that may lead to defects or vulnerabilities. Here are some key points about static testing tools:

1. **Early Detection:** Static testing tools enable the early detection of issues in the software development process, often before the code is even compiled or executed. This can help prevent defects from being introduced into the codebase.

2. **Code Review:** They are often used in code review processes to check for code style violations, coding standards adherence, and common programming errors.

3. **Types of Static Analysis:** Static testing tools use various types of static analysis, such as code reviews, code inspections, and automated code analysis. They can also analyze documents like requirements specifications and design documents.

4. **Common Issues:** These tools can identify issues like syntax errors, logical errors, code smells, security vulnerabilities, and violations of coding standards.

5. **Examples:** Common static testing tools include SonarQube, ESLint (for JavaScript), Checkstyle (for Java), and various code linters.

Dynamic Testing Tools

Dynamic testing tools, on the other hand, are used to evaluate the behavior of a software application while it is running. They involve the execution of the software and the observation of how it performs under different conditions. Here are some key points about dynamic testing tools:

1. **Behavior Evaluation:** Dynamic testing tools assess the software's behavior in real or simulated runtime conditions. They are focused on finding defects and ensuring that the software functions correctly.

2. **Testing Types:** Dynamic testing tools can be used for various types of testing, including functional testing, regression testing, performance testing, load testing, and security testing.

3. Execution of Test Cases: These tools execute predefined test cases or test scripts and compare the actual results with expected results to determine if the software is working as intended.

4. Examples: Common dynamic testing tools include Selenium (for web application testing), JUnit (for unit testing in Java), and Apache JMeter (for load and performance testing).

Factors that needs to be considered before selecting any tool

Selecting the right software testing tool is a critical decision in the softwaredevelopment and testing process.

- It can significantly impact the efficiency and effectiveness of your testingefforts.
- Before selecting a tool, consider the following factors:

1. Testing Requirements: Understand the specific testing needs of your project. Different tools cater to various types of testing, such as functional, performance, security, or compatibility testing. Ensure the tool aligns with your testing requirements.

2. Budget: Consider the cost of the tool, including initial purchase, licensing, maintenance, and any additional costs, such as training or support. Assess whether it fits within your project's budget.

3. Ease of Use: Evaluate the tool's user-friendliness. An intuitive interface and ease of use are important, as they can reduce the learning curve for your team.

4. Integration Capabilities: Ensure that the tool can seamlessly integrate with your existing development and testing tools and frameworks, such as version control systems, continuous integration tools, or issue tracking systems.

5. Supported Technologies: Verify that the tool supports the technologies and platforms used in your project. It should be compatible with your programming languages, frameworks, operating systems, browsers, and databases.

6. Scalability: Consider whether the tool can scale as your project or testing needs grow. Scalability is crucial for larger or long-term projects.

7. Reporting and Analytics: Check the tool's reporting and analytics capabilities. It should provide comprehensive and customizable reports to help you analyze test results effectively.

8. Community and Support: Investigate the tool's user community and available support resources. Active user communities and ample documentation can be valuable when troubleshooting issues.

9. Vendor Reputation: Research the reputation of the tool's vendor. Look for customer reviews, case studies, and references to gauge the vendor's reliability and support.

10. Customization and Extensibility: Assess whether the tool allows customization and extensibility to meet the unique needs of your project. The ability to create custom test scripts or plugins is beneficial.

11. Security and Compliance: If your project involves sensitive data or has specific compliance requirements (e.g., GDPR, HIPAA), ensure that the tool can support security and compliance testing.

12. Maintenance and Updates: Understand the tool's maintenance requirements and update frequency. Frequent updates and good customer support are essential for addressing issues and vulnerabilities.

13. Licensing Model: Determine whether the tool's licensing model (e.g., open- source, freemium, commercial) is suitable for your organization and project.

14. Training and Skill Sets: Assess the availability of training resources and the familiarity of your team with the tool or similar tools. Consider the learning curve for the tool

15. ROI (Return on Investment): Calculate the potential return on investment by using the tool. Consider how much time and cost it can save in the long run compared to manual testing or other tools.

Metrics & Measurement

What is metrics ?

Metrics in software testing refer to quantifiable measures and indicators used to assess various aspects of the testing process and the quality of the software being tested. These metrics help testing teams, project managers, and stakeholders track progress, identify areas for improvement, and make data- driven decisions to ensure the quality and reliability of the software. Software testing metrics can cover a wide range of areas, including test progress, test coverage, defect tracking, and more.

Characteristics of software Metrics:

1. **Quantitative:** Metrics must possess quantitative nature. It means metrics canbe expressed in values.
2. **Understandable:** Metric computation should be easily understood, and themethod of computing metrics should be clearly defined.
3. **Applicability:** Metrics should be applicable in the initial phases of thedevelopment of the software.
4. **Repeatable:** The metric values should be the same when measured repeatedly and consistent in nature.
5. **Economical:** The computation of metrics should be economical.
6. **Language Independent:** Metrics should not depend on any programming language.

What is measurement ?

A measurement is a manifestation of the size, quantity, amount, or dimension of a particular attribute of a product or process. Software measurement is a titrate impute of a characteristic of a software product or the software process. It is an authority within software engineering. The software measurement process is defined and governed by ISO Standard.

Software Measurement Principles:

The software measurement process can be characterized by five activities-

1. **Formulation:** The derivation of software measures and metrics appropriatefor the representation of the software that is being considered.
2. **Collection:** The mechanism used to accumulate data required to derive the formulated metrics.
3. **Analysis:** The computation of metrics and the application of mathematicaltools.
4. **Interpretation:** The evaluation of metrics resulting in insight into the qualityof the representation.
5. **Feedback:** Recommendation derived from the interpretation of productmetrics transmitted to the software team.
6.

Classification of Software Metrics

1. **Product Metrics:** Product metrics are used to evaluate the state of the product, tracing risks and undercover prospective problem areas. The ability of the team to control quality is evaluated. Examples include lines of code, cyclomatic complexity, code coverage, defect density, and code maintainability index.
2. **Process Metrics:** Process metrics pay particular attention to enhancing the long-term process of the team or organization. Examples include effort variance, schedule variance, defect injection rate, and lead time.
3. **Project Metrics:** The project matrix describes the project characteristic and execution process. Examples include effort estimation accuracy, schedule deviation, cost variance, and productivity.
    - Number of software developer
    - Staffing patterns over the life cycle of software
    - Cost and schedule
    - Productivity

Advantages of Software Metrics :

1. Reduction in cost or budget.
2. It helps to identify the particular area for improvising.
3. It helps to increase the product quality.
4. Managing the workloads and teams.
5. Reduction in overall time to produce the product,.
6. It helps to determine the complexity of the code and to test the code with resources.
7. It helps in providing effective planning, controlling and managing of theentire product.

Disadvantages of Software Metrics :

1. It is expensive and difficult to implement the metrics in some cases.
2. It leads to measure the unwanted data which is wastage of time.
3. Sometimes the quality of the product is not met with the expectation.
4. Measuring the incorrect data leads to make wrong decision making.