Test metrics are quantifiable measurements used to evaluate, track, and improve the efficiency, quality, and performance of testing processes. They provide objective data for decision-making, help identify areas for improvement, and are used to report progress to stakeholders. Common examples include defect density, test coverage.

Key aspects of test metrics

Quantifiable data: Metrics turn aspects of the testing process into numerical data that can be analyzed and compared over time or against benchmarks.

Purpose: Their primary goal is to assess the effectiveness of testing activities, track progress, measure the quality of the product, and guide decision-making.

Scope: They are applicable across all stages of the Software Development Life Cycle (SDLC) to ensure a defect-free product.

Reporting: Metrics are used to provide clear, data-driven reports to stakeholders, from the technical team to executive management.

Common types of test metrics

Defect metrics: Measure the number, severity, and density of defects found in the software.

Defect Density: The number of defects found per unit of code (e.g., per thousand lines of code).

Defect Leakage: The number of defects that were not caught during testing and were released to the end-user.

Test coverage metrics: Measure how much of the software's code or functionality has been tested.

Test Coverage: The percentage of total code that is executed by a test suite.

Test Execution Coverage: The percentage of planned test cases that have been executed.

Test effort metrics: Focus on the resources, time, and cost associated with testing.

Test Execution Time: The total time taken to execute test cases.

Cost Per Test Case/Cost Per Defect: The cost incurred to run a single test case or to fix a single defect.

Importance of Metrics in Software Testing:

Test metrics are essential in determining the software's quality and performance. Developers may use the right software testing metrics to improve their productivity.

Early Problem Identification: By measuring metrics such as defect density and defect arrival rate, testing teams can spot trends and patterns early in the development process.

Allocation of Resources: Metrics identify regions where testing efforts are most needed, which helps with resource allocation optimization. By ensuring that testing resources are concentrated on important areas, this enhances the strategy for testing as a whole.

Monitoring Progress: Metrics are useful instruments for monitoring the advancement of testing. They offer insight into the quantity of test cases that have been run, their completion rate, and if the testing effort is proceeding according to plan.

Continuous Improvement: Metrics offer input on the testing procedure, which helps to foster a culture of continuous development.

Types of Software Testing Metrics:

Software testing metrics are divided into three categories:

Process Metrics: A project's characteristics and execution are defined by process metrics. These features are critical to the SDLC process's improvement and maintenance (Software Development Life Cycle).

Product Metrics: A product's size, design, performance, quality, and complexity are defined by product metrics. Developers can improve the quality of their software development by utilizing these features.

Project Metrics: Project Metrics are used to assess a project's overall quality. It is used to estimate a project's resources and deliverables, as well as to determine costs, productivity, and flaws.

It is critical to determine the appropriate testing metrics for the process. A few points to keep in mind:

Before creating the metrics, carefully select your target audiences.

Define the aim for which the metrics were created.

Prepare measurements based on the project's specific requirements. Assess the financial gain associated with each statistic.

Match the measurements to the project lifestyle phase for the best results.

Manual Test Metrics: What Are They and How Do They Work?

Manual testing is carried out in a step-by-step manner by quality assurance experts. Test automation frameworks, tools, and software are used to execute tests in automated testing. There are advantages and disadvantages to both human and automated testing. Manual testing is a time-consuming technique, but it allows testers to deal with more complicated circumstances. There are two sorts of manual test metrics:

1. Base Metrics: Analysts collect data throughout the development and execution of test cases to provide base metrics. By generating a project status report, these metrics are sent to test leads and project managers. It is quantified using calculated metrics.

The total number of test cases

The total number of test cases completed.

2. Calculated Metrics: Data from base metrics are used to create calculated metrics. The test lead collects this information and transforms it into more useful information for tracking project progress at the module, tester, and other levels. It's an important aspect of the SDLC since it allows developers to make critical software changes.

Formula for Test Metrics:

Percentage test cases executed = (No of test cases executed / Total no of test cases written) x 100

1. Test Case Effectiveness:

Test Case Effectiveness = (Number of defects detected / Number of test cases run) x 100

2. Passed Test Cases Percentage: Test Cases that Passed Coverage is a metric that indicates the percentage of test cases that pass.

Passed Test Cases Percentage = (Total number of tests ran / Total number of tests executed) x 100

3. Failed Test Cases Percentage: This metric measures the proportion of all failed test cases.

Failed Test Cases Percentage = (Total number of failed test cases / Total number of tests executed) x 100

4. Blocked Test Cases Percentage: During the software testing process, this parameter determines the percentage of test cases that are blocked.

Blocked Test Cases Percentage = (Total number of blocked tests / Total number of tests executed) x 100

5. Fixed Defects Percentage: Using this measure, the team may determine the percentage of defects that have been fixed.

Fixed Defects Percentage = (Total number of flaws fixed / Number of defects reported) x 100

Test Estimation:

Test estimation is the process of predicting the time, effort, and resources needed for software testing activities. It is a crucial project management activity that helps in planning schedules, budgeting, and allocating resources effectively. Effective estimation involves breaking down tasks, considering factors like project complexity and staff skills, and continuously refining estimates throughout the project lifecycle.

What test estimation includes

Effort: The total hours (or other units like story points) required for testing tasks.

Time: The duration for completing testing, broken down into working days or calendar time, considering deadlines.

Cost: The budget needed for testing, which includes costs for resources, tools, and infrastructure.

Key steps for effective estimation:

Work Breakdown: Decompose the entire testing process into smaller, manageable tasks, activities, and levels.

Task Identification: Define specific activities within each phase, such as test planning, analysis, design, and execution.

Estimation: Assign effort, time, and cost estimates to each individual task and then aggregate them to get an overall estimate.

Factor Analysis: Consider various factors that influence the estimate, including the complexity of the product, development model, and team competence and experience.

Buffer Inclusion: Add contingency time to account for unexpected issues or changes.

Continuous Review: Regularly compare your estimates against actual results from previous projects to improve accuracy for future work.

Communication: Clearly communicate your assumptions, the basis of your estimates, and potential trade-offs to stakeholders.

Test Estimation Best Practices:

Resource planning in estimation: Resource planning and allocation play a key part in estimating the delivery of the project and meeting deadlines. The availability of resources will not only deliver updates to key stakeholders but also assure that we set our estimations right.

Taking references from past projects: Some projects may resemble each other with respect to their domain testing or open-source frameworks or the same client. Any difficulties faced in the past should be noted and care must be taken so that they do not recur in the future.

Keeping some buffer time: Unforeseen circumstances can happen which include the departure of critical team members or unplanned long leaves. This could delay the scheduled timeline of the project and care must be taken to preserve some buffer time.

Sticking to the Estimation: Whatever estimation we have made, we should try to adhere to that irrespective of its correctness or feasibility. Unless any major change is encountered, we should try not to modify the estimate and stick to it.

Considering the Bug Cycle: The test estimation has the bug cycle included in it. The encountering of bugs causes a delay in the actual test cycle and hence the number of estimated days increases.

Scope of Project: Knowing our project objective is important as it helps to estimate between small and large projects and our project delivery depends on it. Large projects have lots of test scripts, test data, and test documents. Therefore, knowing our objective of testing and planning of deliverables plays a key role in knowing the project scope.

Load Testing Plan: We need to know the estimated time if there is a requirement to perform Load Testing and hence plan the estimates accordingly.

Parallel Testing: The scope of parallel testing needs to be known i.e. whether we have the previous versions of the same product for comparing the output. If there is a chance, the testing process becomes easier and product estimation can be done in a better way.

Risk Based Testing:

Risk-based testing is a software testing method that prioritizes testing efforts based on the potential risk of a feature's failure. Instead of treating all tests equally, it focuses limited time and resources on the areas that are most likely to have defects or where failure would have a significant impact on users or the business. This approach is especially beneficial for complex projects, **tight timelines**, or when dealing with high-risk systems.

Key principles:

Risk assessment: The process starts with identifying and assessing risks, such as the likelihood and impact of defects. Risks can be based on factors like the criticality of the business function, frequency of use, or a history of defects.

Prioritization: Based on the risk assessment, test cases are prioritized. High-risk areas receive more thorough testing, while lower-risk areas may receive less testing.

Resource allocation: Testing resources are strategically allocated to ensure the most critical functions are tested first.

Continuous process: Risk-based testing is not a one-time activity; the test plan and strategy should be revisited and adjusted as the project progresses and new risks emerge.

Benefits:

Optimizes resources: It ensures that the most critical functionalities are tested thoroughly, especially when time and budget are limited.

Improves quality: By focusing on critical areas, it helps improve the overall reliability and quality of the software, leading to higher user satisfaction.

Reduces risk of failure: It proactively mitigates risks by focusing on high-impact areas, reducing the chances of significant financial or data loss due to bugs.

Increases efficiency: It streamlines the testing process by reducing redundant tests and focusing effort where it is most needed

When To Implement Risk-Based Testing?

Risk-based testing approach is implemented in scenarios where-

- There is time/resource or budget constraints. For example- A hotfix to be deployed in production.
- When a proof of concept is being implemented.
- When the team is new to the technology platform or to the application under test.
- When testing in Incremental models or Iterative models.
- Security testing is done in Cloud computing environments.