

## **UNIT - V**

### **CHAPTER IX**

#### 9.1 Software Test Automation

### **CHAPTER X**

#### 10.1 Test Metrics and Measurements.

#### 10.2 Types of Metrics

#### 10.3 Check your progress

## 9.1 SOFTWARE TEST AUTOMATION

### Introduction

Developing software to test the software is called test automation. Test automation can help address several problems.

- Automation saves time as software can execute test case faster than human do.
- Test automation can free the test engineers from mundane tasks and make them focus on more creative tasks.
- Automated tests can be more reliable
- Automation helps in immediate testing
- Automation can protect an organization against attrition of test engineers.
- Test automation opens up opportunities for better utilization of global resources.
- Certain types of testing cannot be executed without automation
- Automation means end-to-end, not test execution alone.

Automation should have scripts that produce test data to maximize coverage of permutations and combinations of inputs and expected output for result comparison. They are called *test data generators*. The automation script should be able to map the error patterns dynamically to conclude the result. The error pattern mapping is done not only to conclude the result of a test, but also point out the root cause.

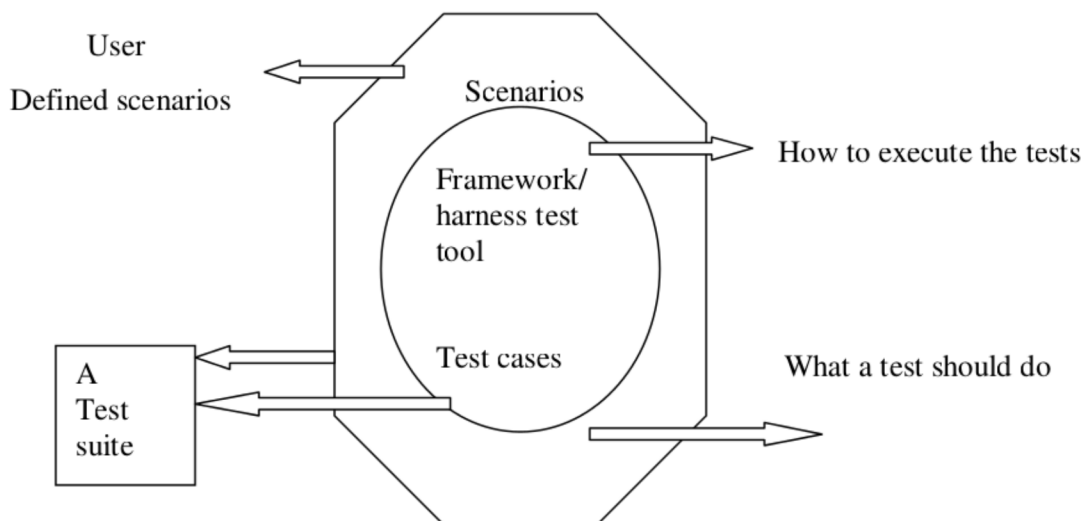
### Terms used in automation

A test case is a set of sequential steps to execute a test operating on a set of predefined inputs to produce certain expected outputs. There are two types of test cases namely automated and manual. Test case in this chapter refers to automated test cases. A test case can be documented as a set of simple steps, or it could be an assertion statement or a set of assertions. An example of assertion is “Opening a file, which is already opened should fail.” The following table describes some test cases for the log in example, on how the log in can be tested for different types of testing.

S.No.	Test cases for testing	Belongs to what type of testing
1.	Check whether login works	Functionality
2.	Repeat log in operation in a loop for 48 hours	Reliability
3.	Perform log in from 10000 clients	Load/Stress testing
4.	Measure time taken for log in operations In different conditions	Performance
5.	Run log in operation from a machine running Japanese language	Internalization

**Table- Same test case being used for different types of testing**

In the above table the how portion of the test case is called *scenarios*. What an operation has to do is a product specific feature and how they are to be run is a *framework*-specific requirement. When a set of test cases is combined and associated with a set of scenarios, they are called “*test suite*”.



**Fig. Framework for test automation**

## **Skills Needed for Automation**

The automation of testing is broadly classified into three generations.

### **First generation – record and playback**

Record and playback avoids the repetitive nature of executing tests. Almost all the test tools available in the market have the record and playback feature. A test engineer records the sequence of actions by keyboard characters or mouse clicks and those recorded scripts are played back later, in the same order as they were recorded. When there is frequent change, the record and playback generation of test automation tools may not be very effective.

### **Second generation – data – driven**

This method helps in developing test scripts that generates the set of input conditions and corresponding expected output. This enables the tests to be repeated for different input and output conditions. This generation of automation focuses on input and output conditions using the black box testing approach.

### **Third generation action driven**

This technique enables a layman to create automated tests; there are no input and expected output condition required for running the tests. All action that appear on application are automatically tested based on a generic set of controls defined for automation. Input and output condition are automatically generated and used the scenarios for test execution can be dynamically changed using the test framework that available in this approach of automation hence automation in the third generation involves two major aspects “test case automation” and “frame work design”.

## **What to Automate, Scope of Automation**



The specific requirements can vary from product to product, from situation to situation, from time to time. The following gives some generic tips for identifying the scope of automation.

### **Identifying the types of testing amenable to automation**

#### **Stress, reliability, scalability, and performance testing**

These types of testing require the test cases to be run from a large number of different machines for an extended period of time, such as 24 hours, 48 hours, and so on. Test cases belonging to these testing types become the first candidates for automation.

#### **Regression tests**

Regression tests are repetitive in nature. Given the repetitive nature of the test cases, automation will save significant time and effort in the long run.

#### **Functional tests**

These kinds of tests may require a complex set up and thus required specialized skill, which may not be available on an ongoing basis. Automating these once, using the expert skill tests, can enable using less-skilled people to run these tests on an ongoing basis.

#### **Automating areas less prone to change**

User interfaces normally go through significant changes during a project. To avoid rework on automated test cases, proper analysis has to be done to find out the areas of changes to user interfaces, and automate only those areas that will go through relatively less change. The non-user interface portions of the product can be automated first. This enables the non-GUI portions of the automation to be reused even when GUI goes through changes.

#### **Automate tests that pertain to standards**

One of the tests that products may have to undergo is compliance to standards. For example, a product providing a JDBC interface should satisfy the standard JDBC tests. Automating for standards provides a dual advantage. Test suites developed for standards are not only used for product testing but can also be sold as test tools for the market. Testing for standards have certain legal requirements. To certify the software, a test suite is developed and handed over to different companies. This is called “certification testing” and requires perfectly compliant results every time the tests are executed.

### **Management aspects in automation**

Prior to starting automation, adequate effort has to be spent to obtain management commitment. The automated test cases need to be maintained till the product reaches obsolescence. Since automation involves effort over an extended period of time, management permissions are only given in phases and part by part. It is important to automate the critical and basic functionalities of a product first. To achieve this, all test cases need to be prioritized as high, medium, and low, based on customer expectations. Automation should start from high priority and then over medium and low-priority requirements.

### **Design and Architecture for Automation**

Design and architecture is an important aspect of automation. As in product development, the design has to represent all requirements in modules and in the interactions between modules.

In integration testing both internal interfaces and external interfaces have to be captured by design and architecture. Architecture for test automation involves two major heads: a test infrastructure that covers a test case database and a defect database or defect repository. Using this infrastructure, the test framework provides a backbone that ties the selection and execution of test cases.

### **External modules**

There are two modules that are external modules to automation – TCDB and defect DB. Manual test cases do not need any interaction between the framework and TCDB. Test engineers submit the defects for manual test cases. For automated test cases, the framework can automatically submit the defects to the defect DB during execution. These external modules can be accessed by any module in automation framework.

### **Scenario and configuration file modules**

Scenarios are information on “how to execute a particular test case“. A configuration file contains a set of variables that are used in automation. A configuration file is important for running the test cases for various execution conditions and for running the tests for various input and output conditions and states. The values of variables in this configuration file can be changed dynamically to achieve different execution input, output and state conditions.

### **Test cases and test framework modules**

Test case is an object for execution for other modules in the architecture and does not represent any interaction by itself. A test framework is a module that combines “what to execute” and “ how they have to be executed.” The test framework is considered the core of automation design. It can be developed by the organization internally or can be bought from the vendor.

### **Tools and results modules**

When a test framework performs its operations, there are a set of tools that may be required. For example, when test cases are stored as source code files in TCDB, they need to be extracted and compiled by build tools. In order to run the compiled code, certain runtime tools and utilities may be required.

The results that come out of the test must be stored for future analysis. The history of all the previous tests run should be recorded and kept as archives. This results help the test engineer to execute the test cases compared with the previous test run. The audit of all tests that are run and the related information are stored in the module of automation. This can also help in selecting test cases for regression runs.

### **Report generator and reports /metrics modules**

Once the results of a test run are available, the next step is to prepare the test reports and metrics. Preparing reports is a complex work and hence it should be part of the automation design. The periodicity of the reports is different, such as daily, weekly, monthly, and milestone reports. Having reports of different levels of detail can address the needs of multiple constituents and thus provide significant returns.

The module that takes the necessary inputs and prepares a formatted report is called a *report generator*. Once the results are available, the report generator can generate metrics. All the reports and metrics that are generated are stored in the reports/metrics module of automation for future use and analysis.

### **Generic Requirements for Test Tool/Framework**

In the previous section, we described a generic framework for test automation. This section present a detailed criteria that such a framework and its usage should satisfy.

1. No hard coding in the test suite.
2. Test case/suite expandability.
3. Reuse of code for different types of testing, test cases.
4. Automatic setup and cleanup.
5. Independent test cases.
6. Test case dependency
7. Insulating test cases during execution
8. Coding standards and directory structure.

9. Selective execution of test cases.
10. Random execution of test cases.
11. Parallel execution of test cases.
12. Looping the test cases
13. Grouping of test scenarios
14. Test case execution based on previous results.
15. Remote execution of test cases.
16. Automatic archival of test data.
17. Reporting scheme.
18. Independent of languages
19. Probability to different platforms.

### **Process Model for Automation**

The work on automation can go simultaneously with product development and can overlap with multiple releases of the product. One specific requirement for automation is that the delivery of the automated tests should be done before the test execution phase so that the deliverables from automation effort can be utilized for the current release of the product.

Test automation life cycle activities bear a strong similarity to product development activities. Just as product requirements need to be gathered on the product side, automation requirements too need to be gathered. Similarly, just as product planning, design and coding are done, so also during test automation are automation planning, design and coding.

After introducing testing activities for both the product and automation, the above figure includes two parallel sets of activities for development and testing separately. When they are put together, it becomes a “W” model. Hence for a product development involving automation, it will be a good choice to follow the W model to ensure that the quality of the product as well as the test suite developed meet the expected quality norms.

## **Selecting a test tool**

Having identified the requirements of what to automate, a related question is the choice of an appropriate tool for automation. Selecting the test tool is an important aspect of test automation for several reasons given below:

1. Free tools are not well supported and get phased out soon.
2. Developing in-house tools take time.
3. Test tools sold by vendors are expensive.
4. Test tools require strong training.
5. Test tools generally do not meet all the requirements for automation.
6. Not all test tools run on all platform.

For all the above strong reasons, adequate focus needs to be provided for selecting the right tool for automation.

## **Criteria for selecting test tools**

In the previous section, we looked at some reasons for evaluating the test tools and how requirements gathering will help. This will change according to context and are different for different companies and products. We will now look into the broad categories for classifying the criteria. The categories are

1. Meeting requirements
2. Technology expectations
3. Training/skills and
4. Management aspects.

## **Meeting requirements**

Firstly, there are plenty of tools available in the market, but they do not meet all the requirements of a given product. Evaluating different tools for different requirements involves significant effort, money and time.

Secondly, test tools are usually one generation behind and may not provide backward or forward compatibility. Thirdly, test tools may not go through the same amount of evaluation for new requirements.

Finally, a number of test tools cannot differentiate between a product failure and a test failure. So the test tool must have some intelligence to proactively find out the changes that happened in the product and accordingly analyze the results.

### **Technology expectations**

- Extensibility and customization are important expectations of a test tool.
- A good number of test tools require their libraries to be linked with product binaries.
- Test tools are not 100% cross platform. When there is an impact analysis of the product on the network, the first suspect is the test tool and it is uninstalled when such analysis starts.

### **Training skills**

While test tools require plenty of training, very few vendors provide the training to the required level. Test tools expect the users to learn new language/scripts and may not use standard languages/scripts. This increases skill requirements for automation and increases the need for a learning curve inside the organization.

### **Management aspects**

- Test tools requires system upgrades.
- Migration to other test tools difficult
- Deploying tool requires huge planning and effort.

### **Steps for tool selection and deployment**

1. Identify your test suite requirements among the generic requirements discussed. Add other requirements if any.
2. Make sure experiences discussed in previous sections are taken care of.
3. Collect the experiences of other organizations which used similar test tools.
4. Keep a checklist of questions to be asked to the vendors on cost/effort/support.
5. Identify list of tools that meet the above requirements.
6. Evaluate and shortlist one/set of tools and train all test developers on the tool.
7. Deploy the tool across the teams after training all potential users of the tool.

### **Challenges in Automation**

The most important challenge of automation is the management commitment. Automation takes time and effort and pays off in the long run. Management should have patience and persist with automation. Successful test automation endeavors are characterized unflinching management commitment, a clear vision of goals that track progress with respect to the long-term vision.



## CHAPTER X

### 10.1 TEST METRICS AND MEASUREMENTS

#### What are metrics and measurements

Metrics derive information from raw data with a view to help in decision making. Some of the areas that such information would shed light on are

- ✓ Relationship between the data points
- ✓ Any cause and effect correlation between the observed data points.
- ✓ Any pointers to how the data can be used for future planning and continuous improvements

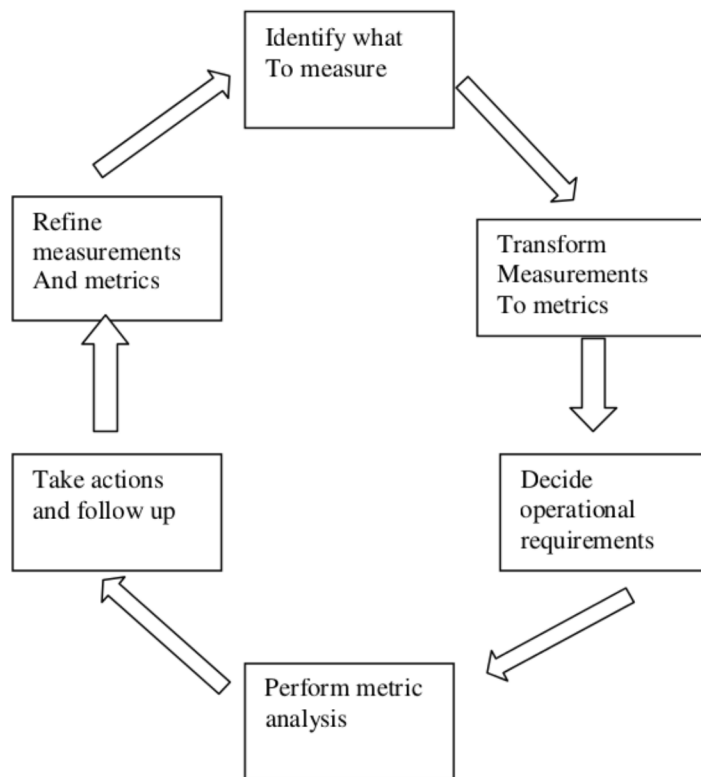
Metrics are thus derived from measurement using appropriate formulas or calculation. Obviously the same set measurement can help produce different set of metrics of interest to different people.

From the above discussion it is obvious that in order that a project performance be tracked and its progress monitored effectively,

- The right parameters must be measured; the parameters may pertain to product or to process
- The right analysis must be done on the data measured to draw within a project or organization.
- The result of the analysis must be presented in an appropriate form to the stakeholders to enable them to make the right decisions on improving product or process quality

*Effort* is the actual time that is spent on a particular activity or a phase. Elapsed days is the difference between the start of an activity and the completion of the activity.

Collecting and analyzing metrics involves effort and several steps. This is depicted in the following figure.



**Figure. Steps in a metrics program**

The first step involved in a metrics programme is to decide what measurement are important and collect data accordingly. The effort spent on testing number of defect and number of test cases are some examples of measurement Depending on what the data is used for the granularity of measurement will vary.

While deciding what to measure the following aspects need to be kept in mind.

1. What is measured should be of relevance to what we are trying to achieve. For testing functions we would obviously be interested in the effort spent on testing number of test cases number of defects reported from test cases and so on.
2. The entities measured should be natural and should not involve too many overheads for measurement. If there are too many overheads in making the measurements do not follows naturally from the actual work being done then the people who supply the data may resist giving the measurement data.
3. What is measured should be at the right level of granularity to satisfy the objective for which the measurement is being made.

An approach involved in getting the granular detail is called *data drilling*

It is important to provide as much granularity in measurement as possible. A set of measurement can be combined to generate metrics. An example question involving multiple measurements is “How many test cases produced the 40 defect in date migration involving different schema?” There are two measurements involved in this question the number of test cases and the number of defect .Hence the second step involved in metrics collection is defining how to combine data points or measurements to provide meaningful metrics. A particular metric can use one or more measurements. The operational requirement for a metrics plan should lay down not only the periodicity but also other operational issues such as who should collect measurements, who should receive the analysis, and so on. The final step involved in a metrics plan is to take necessary action and follow up on the action.

### **Why Metrics in Testing**

Metrics are needed to know test case execution productivity and to estimate test completion date.

Days needed to complete testing = total test cases yet to be executed /

Test case execution productivity

The defect fixing trend collected over a period of time gives another estimate of the defect-fixing capability of the team.

Total days needed for defect fixes = (Outstanding defects yet to be fixed + Defects That can be found in future test cycles) / Defect fixing capability.

Hence, metrics helps in estimating the total days needed for fixing defects. Once the time needed for testing and the time for defect fixing are known, the release date can be estimated.

Days needed for release = Max (days needed for testing, days needed for defect fixes ).

The defect fixes may arrive after the regular test cycles are completed. These defect fixes have to be verified by regression testing before the product can be released. Hence the formula for days needed for release is to be modified as follows:

Days needed for release = Max [ Days needed for testing, [ Days needed for defect fixes+ Days needed for regressing outstanding defect fixes]

The idea of discussing the formula here is to explain that metrics are important and help in arriving at the release date for the product. Metrics are not only used for reactive activities. Metrics and their analysis help in preventing the defects proactively, thereby saving cost and effort. Metrics are used in resource management to identify the right size of product development teams.

To summarize, metrics in testing help in identifying

- When to make the release.
- What to release
- Whether the product is being released with known quality.

## 10.2 Types of Metrics

Metrics can be classified into different types based on what they measure and what area they focus on. Metrics can be classified as *product metrics* and *process metrics*.

Product metrics can be further classified as

**Project metrics** – a set of metrics that indicates how the project is planned and executed.

**Progress metrics** – a set of metrics that tracks how the activities of the project are progressing.

**Productivity metrics** – a set of metrics that takes up into account various productivity numbers that can be collected and used for planning and tracking testing activities.

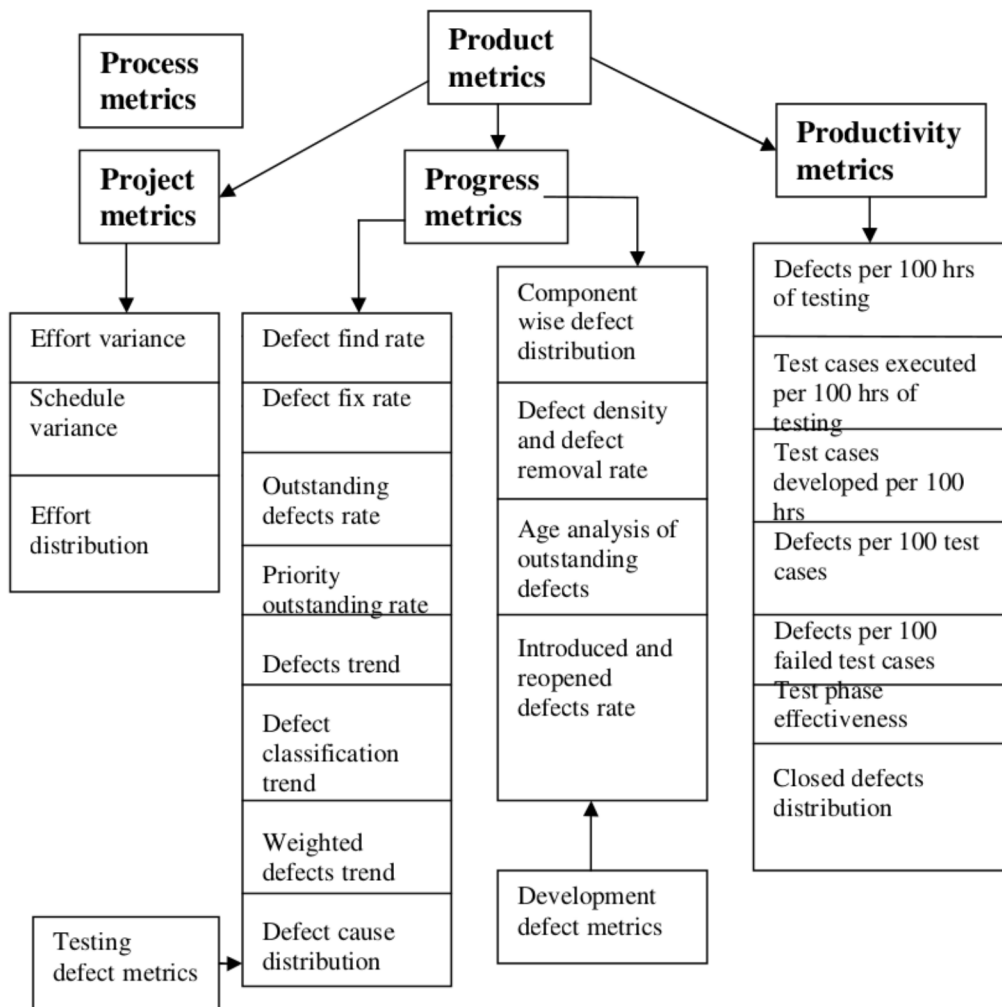
### Project Metrics

A typical project starts with requirements gathering and ends with product release. All the phases that fall in between these points need to be planned and tracked. The project scope gets translated to size estimates, which specify the quantum of work to be done. This size estimate gets translated to effort estimate for each of the phases and activities by using the available productivity data available. This initial effort is called *baselined effort*.

As the project progresses and if the scope of the project changes then the effort estimates are re-evaluated again and this re-evaluated effort estimate is called *revised effort*.

### **Effort variance ( planned vs actual )**

If there is substantial difference between the baselined and revised effort, it points to incorrect initial estimation. Calculating effort variance for each of the phases provides a quantitative measure of the relative difference between the revised and actual efforts.



**Fig. Types of metrics**

<b>Effort</b>	<b>Req</b>	<b>Design</b>	<b>Coding</b>	<b>Testing</b>	<b>Doc</b>	<b>Defect fixing</b>
<b>Variance %</b>	7.1	8.7	5	0	40	15

**Table. Sample variance percentage by phase.**

$$\text{Variance \%} = [(\text{Actual effort} - \text{Revised estimate}) / \text{Revised estimate}] * 100$$

A variance more than 5% in any of the SDLC phase indicates the scope for improvement in the estimation. The variance can be negative also. A negative variance is an indication of an over estimate. These variance numbers along with analysis can help in better estimation for the next release or the next revised estimation cycle.

**Schedule variance ( planned vs actual )**

Schedule variance, like effort variance, is the deviation of the actual schedule from the estimated schedule. There is one difference though. Depending on the SDLC model used by the project, several phases could be active at the same time. Further the different phases in SDLC are interrelated and could share the same set of individuals. Because of all these complexities involved, schedule variance is calculated only at the overall project level, at specific milestones, not with respect to each of the SDLC phases.

Effort and schedule variance have to be analyzed in totality, not in isolation. This is because while effort is a major driver of the cost, schedule determines how best a product can exploit market opportunities. Variance can be classified into negative variance, zero



variance, acceptable variance and unacceptable variance. Generally 0-5% is considered as acceptable variance.

### **Effort distribution across phases**

Variance calculation helps in finding out whether commitments are met on time and whether the estimation method works well. The distribution percentage across the different phases can be estimated at the time of planning and these can be compared with the actual at the time of release for getting a comfort feeling on the release and estimation methods.

Mature organizations spend at least 10-15% of the total effort in requirements and approximately the same effort in the design phase. The effort percentage for testing depends on the type of release and amount of change to the existing code base and functionality. Typically, organizations spend about 20 – 50% of their total effort in testing.

### **Progress Metrics**

The number of defects that are found in the product is one of the main indicators of quality. Hence, we will look at progress metrics that reflect the defects of a product. Defects get detected by the testing team and get fixed by the development team. Based on this, defect metrics are further classified into *test defect metrics* and *development defect metrics*.

How many defects have already been found and how many more defects may get unearthed are two parameters that determine product quality and its assessment, the progress of testing has to be understood. If only 50% of testing is complete and if 100



defects are found, then, assuming that the defects are uniformly distributed over the product, another 80-100 defects can be estimated as residual defects.

### 1. Test defect metrics

The next set of metrics help us understand how the defects that are found can be used to improve testing and product quality. Not all defects are equal in impact or importance. Some organizations classify defects by assigning a defect priority. The priority of a defect provides a management perspective for the order of defect fixes. Some organization use defect severity levels, they provide the test team a perspective of the impact of that defect in product functionality. Since different organizations use different methods of defining priorities and severities, a common set of defect definitions and classification are provided in the table given below.

#### Defect find rate

When tracking and plotting the total number of defects found in the product at regular intervals, from beginning to end of a product development cycle, it may show a pattern for defect arrival. For a product to be fit for release, not only is such a pattern of defect arrival in a particular duration should be kept at a bare minimum number. A bell curve along with minimum number of defects found in the last few days indicate that the release quality of the product is likely to be good.

<b>Defect classification</b>	<b>What it means</b>
Extreme	 Product crashes or unusable  Needs to be fixed immediately

Critical	<ul style="list-style-type: none"> <li><input type="checkbox"/> Basic functionality of the product not working</li> <li><input type="checkbox"/> Needs to be fixed before next test cycle starts</li> </ul>
Important	<ul style="list-style-type: none"> <li><input type="checkbox"/> Extended functionality of the product not working</li> <li><input type="checkbox"/> Does not affect the progress of testing</li> </ul>
Minor	<ul style="list-style-type: none"> <li><input type="checkbox"/> Product behaves differently</li> <li><input type="checkbox"/> No impact on the test team or customers</li> <li><input type="checkbox"/> Fix it when time permits</li> </ul>
Cosmetic	<ul style="list-style-type: none"> <li><input type="checkbox"/> Minor irritant</li> </ul> <p>Need not be fixed for this release</p>

**Table. A common defect definition and classification**

### **Defect fix rate**

If the goal of testing is to find defects as early as possible, it is natural to expect that the goal of development should be to fix defects as soon as they arrive. If the defect fixing curve is in line with defect arrival a “bell curve” will be the result again. There is a reason why defect fixing rate should be same as defect arrival rate. As discussed in the regression testing, when defects are fixed in the product, it opens the door for the introduction of new defects. Hence, it is a good idea to fix the defects early and test those defect fixes thoroughly to find our all introduced defects. If this principle is not followed, defects introduced by the defect fixes may come up for testing just before the release and end up in surfacing of new defects.

### **Outstanding defects rate**

The number of defects outstanding in the product is calculated by subtracting the total defects fixed from the total defects found in the product. If the defect fixing pattern is constant like a straight line, the outstanding defects will result in a bell curve again. If the defect-fixing pattern matches the arrival rate, then the outstanding defects curve will look like a straight line. However it is not possible to fix all defects when the arrival rate is at the top end of the bell curve. Hence, the outstanding defect rate results in a ball curve in many projects. When testing is in progress, the outstanding defects should be kept very close to zero so that the development team's bandwidth is available to analyze and fix the issues soon after they arrive.

### **Priority outstanding rate**

Sometimes the defects that are coming out of testing may be very critical and may take enormous effort to fix and to test. Hence, it is important to look at how many serious issues are being uncovered in the product. The modification to the outstanding defects rate curve by plotting only the high priority defects is called priority outstanding defects. The priority outstanding defects correspond to extreme and critical classification of defects. Some organizations include important defects also in priority outstanding defects.

The effectiveness of analysis increases when several perspectives of find rate, fix rate, outstanding, and priority outstanding defects are combined. There are different defect trends like defect trend, defect classification trend and weighted defects trend.

### **Development defect metrics**

In this section we will look how metrics can be used to improve the development activities. The defect metrics that directly help in improving development activities are discussed in this section and are termed as development defect metrics. While defect metrics focus on the number of defects, development defect metrics try to map those defects to different components of the product and to some of the parameters of development such as lines of code.

### **Component-wise defect distribution**

While it is important to count the number of defects in the product, for development it is important to map them to different components of the product so that they can be assigned to the appropriate developer to fix those defects. The project manager in charge of development maintains a module ownership list where all product modules and owners are listed. Based on the number of defects existing in each of the modules, the project manager assigns resources accordingly.

### **Defect density and defect removal rate**

A good quality product can have a long lifetime before becoming obsolete. The lifetime of the product depends on its quality, over the different releases it goes through. One of the metrics that correlates source code and defects is defect density. This metric maps the defects in the product with the volume of code that is produced for the product.

There are several standard formulae for calculating defect density. Of these, defects per KLOC is the most practical and easy metric to calculate and plot. KLOC stands for kilo lines of code, every 1000 lines of executable statements in the product is counted as one KLOC.

The metric compares the defects per KLOC of the current release with previous releases of the product. There are several variants of this metric to make it relevant to releases, and one of them is calculating AMD ( added, modified, deleted code) to find out how a particular release affects product quality.

$$\text{Defects per KLOC} = \left( \frac{\text{Total defects found in the product}}{\text{Total executable AMD lines of code in KLOC}} \right)$$

Defects per KLOC can be used as a release criteria as well as a product quality indicator with respect to code and defects. Defects found by the testing team have to be fixed by the development team. The ultimate quality of the product depends both on development and testing activities and there is a need for a metric to analyze both the development and the testing phases together and map them to release. The defect removal rate is used for the purpose.

The formula for calculating the defect removal rate is

$$\left( \frac{\text{Defects found by verification activities} + \text{Defects found in unit testing}}{\text{Defects found by test teams}} \right) * 100$$

The above formula helps in finding the efficiency of verification activities and unit testing which are normally responsibilities of the development team and compare them to the defects found by the testing teams. These metrics are tracked over various release to study in-release-on-release trends in the verification /quality assurance activities.

### **Age analysis of outstanding defects**

Age here means those defects that have been waiting to be fixed for a long time. Some defects that are difficult to be fixed or require significant effort may get postponed for a

longer duration. Hence, the age of a defect in a way represents the complexity of the defect fix needed. Given the complexity and time involved in fixing those defects, they need to be tracked closely else they may get postponed close to release which may even delay the release. A method to track such defects is called age analysis of outstanding defects.

### **Productivity Metrics**

Productivity metrics combine several measurements and parameters with effort spent on the product. They help in finding out the capability of the team as well as for other purposes, such as

- a. Estimating for the new release
- b. Finding out how well the team is progressing, understanding, the reasons for variation in results.
- c. Estimating the number of defects that can be found.
- d. Estimating release date and quality.
- e. Estimating the cost involved in the release.

### **Defects per 100 hours of testing**

Program testing can only prove the presence of defects, never their absence. Hence, it is reasonable to conclude that there is no end to testing and more testing may reveal more new defects. But here may be a point of diminishing returns when further testing may not reveal any defects. If incoming defects in the product are reducing, it may mean various things.

1. Testing is not effective.
2. The quality of the product is improving

3. Effort spent in testing is falling.

Defects per 100 hours of testing = ( Total defects found in the product for a period/ Total hours spent to get those defects) \* 100

### **Test cases executed per 100 hours of testing**

The number of test cases executed by the test team for a particular duration depends on team productivity and quality of product. The team productivity has to be calculated accurately so that it can be tracked for the current release and be used to estimate the next release of the product.

Test cases executed per 100 hours of testing = ( Total test cases executed for a period / total hours spent in test execution) \* 100

### **Test cases developed per 100 hours of testing**

Both manual and automating test cases require estimating and tracking of productivity numbers. In a product scenario not all test cases are written afresh for every release New test cases are added to address new functionality and for testing features that were not tested earlier Existing test cases are modified to reflect changes in the product . Some test cases are deleted changers in the product, Some test cases are deleted if they are no longer useful or if corresponding features are removed from the product features are removed from the product, Hence the formula for test cases developed uses the count corresponding to added modified and deleted test cases,

Test cases developed per 100 hours of testing= Total test cases developed for a period / total hours spent in test case development 100



## **Defect per 100 Test Cases**

Since the goal of testing is find out as many defects as possible, it is appropriate to measure the defect yield of test that is how many defects get uncovered during testing. This is a function of two parameters one the effectiveness of the tests that are capable of uncovering defects. The ability of test case to uncover defect depend on how well the test cases are designed and developed. But in a typical product scenario not all test cases are executed for every test cycle, Hence it is better to select test cases that produce defect. A measure that quantifies these two parameters is defect per 100 test cases. Yet another parameter that influences this metric is the quality of product, If product quality is poor, it produces more defects per 100 test cases compared to a good quality product. The formula used for calculating this metric is

$$\text{Defects per 100 test cases} = (\text{Total defect found for a period} / \text{Total test cases executed for the same period}) * 100$$

## **Defects per 100 Failed Test Cases**

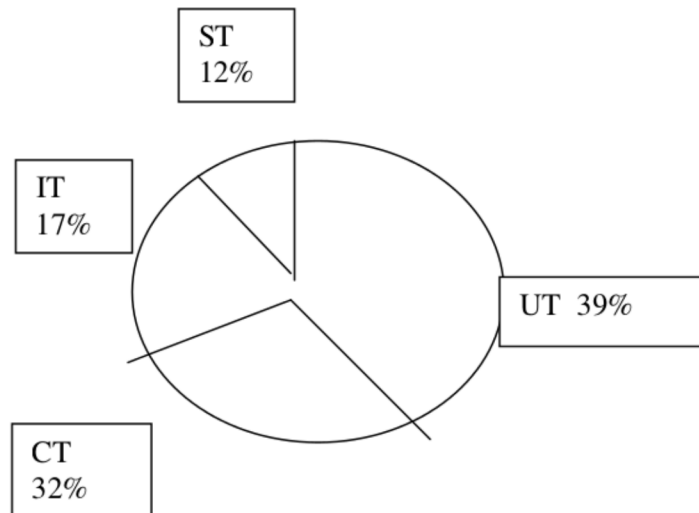
Defect per 100 failed test cases is a good measure to find out how granular the test cases are it indicates

- How many test cases need to be executed when a defect is fixed
- What defect need to be fixed so that an acceptable number of test cases reach the pass state and
- How the fail rate of test cases and defect affect each other for release readiness analysis.

DEFECT PER 100 FAILED TEST CASES = (TOTAL DEFECT FOUND FOR A PERIOD/ TOTAL TEST CASES FAILED DUE TO THOSE DEEFECTS)\* 100

### Test phase effectiveness

As per the principles of testing, testing is not the job of testers alone. Developers perform unit testing and there could be multiple testing teams performing component, integration and system testing phases. The idea of testing is to find defects early in the cycle and in the early phases of testing. As testing is performed by various teams with the objective of finding defects early at various phases, a metric needed to compare the defects filed by each of the phases in testing. The defects found in various phases such as unit testing(UT), component testing(CT), integration testing(IT), and system testing are plotted and analyzed.



In the chart given, the total defects found by each test phase is plotted. The following few observations can be made.

1. A good proportion of defects were found in the early phases of testing( UT and CT).
2. Product quality improved from phase to phase

### **Closed defect distribution**

The objective of testing is not only to find defects. The testing team also has the objective to ensure that all defects found through testing are fixed so that the customer gets the benefit of testing and the product quality improves the testing team has to track the defects and analyze how they are closed.

### **Release Metrics**

The decision to release a product would need to consider several perspectives and metrics. All the metrics that were discussed in the previous section need to be considered in totality for making the release decision. The following table gives some of the perspectives and some sample guidelines needed for release analysis.

Metric	Perspectives to be considered	Guidelines
Test cases executed	Execution %  Pass %	All 100% of test cases to be executed  Test cases passed should be minimum 98%
Effort distribution	Adequate effort has been spent on all phases	15-20% effort spent each on requirements, design, and testing phases.
Defect find rate	Defect trend	Defect arrival trend showing bell curve  Incoming defects close to zero in the last week
Defect fix rate	Defect fix trend	Defect fixing trend matching arrival trend
Outstanding defects trend	Outstanding defects	Outstanding defects trend showing downward trend  Close to zero outstanding defects in the last few weeks prior to release.

### **10.3 Check your progress**

1. Explain the framework of test automation
2. What is software metrics? Explain the types of metrics.
3. What is defect fix rate? Explain.