CHAPTER 7

Software Test Metrics

7.1 Introduction

Software testing has taken a new dimension in the recent past due to increasing complexity and functionality of software. In critical applications, the effort spent on testing can be as high as 35% to 45% of project effort. With the outsourcing of software work in vogue companies are beginning to outsource software testing. Hence it is necessary to define precise processes and process metrics for testing and manage the testing projects objectively. With Testing services getting outsourced, the vendors are getting testing projects where the Cycle Times, Efficiency & Effectiveness of testing, focus on enhanced productivity are extremely challenging and this demand is expected to increase many fold in the coming years. Therefore it is important that the testing projects are quantitatively managed based on data and not on perception. Historically [56,57] testing group’s perspective was to detect as many bugs as possible in the software. Testing as a specialized service is still evolving and is yet to see same level of standardization as development life cycle. However, some trends are evident:

- Increasing trend towards creating an independent testing team aligned with business requirements rather than development priorities
- Testing is being outsourced by Fortune 100 largely to due to time to market, cost and skill advantages of specialist vendors
- Fragmentation of technologies necessitates creation of an independent testing competency. This helps business focus on development rather than playing catch-up with testing methods, tools and skills required for diverse technologies.

- Testing-focused competency building helps retain and develop skills difficult to sustain in a development shop where testing is considered "dead-end" job.

- Off-the-shelf tools have not had success in the market due to their inability to fit into complex environments of Fortune 100 companies. Customization of test environments with some use of off-the-shelf tools is evolving industry direction.

- Component certification and industry based standards are evolving trends in application testing.

- Knowledge creation in testing is rudimentary in software industry today. However, Testing Maturity Models are evolving to measure and build organizational capability for applying experience base to knowledge creation and continuous improvement.

- With increasing complexity of technologies, types, tools and processes of testing, metrics for testing in the form of ROI of Testing, SLAs and Entry/Exit criteria are becoming critical to manage the business of testing.

- Testing services are constantly upgrading themselves and now span from simple execution to strategic consulting services targeted towards defining processes, skills and environments for a complex Fortune 500 environment.

Companies are beginning to rethink outsourcing plans with an aim to completely spurn off activities that are resource intensive, time consuming and do not require strategic inputs from the core development team. Add to these factors like variability in demand and infrastructure and we arrive at software testing – a critical element of the software development life cycle.
that lends itself perfectly and stands to gain the most from offshore outsourcing. With Testing services getting outsourced, the vendors are getting testing projects where the Cycle Times, efficiency & effectiveness of testing, focus on enhanced productivity are extremely challenging and this demand is expected to increase many fold in the coming years. Therefore it is important that we address the issues related to the management of such projects. The key aspect of managing the testing project is collecting and analyzing the data gathered during the course of execution of the project [58,61,62]. Collecting metrics is the best way to know whether a process is under control and within overall tolerances [127,128,129]. The testing process is no different. If you have a large or complex project, you should collect testing metrics to ensure that the process is going well and accomplishing its objectives.

This chapter comes up with the comprehensive list of metrics that can be collected and analyzed for a testing project. Also this chapter defines the various analyses, which can be done on these metrics and what all decision can be taken.

7.2 Objectively Managing Software Testing Projects

The need to measure returns from testing and testing services has increased with increasing investments in complex technology environments and greater outsourcing. Customers who offload testing work to the vendors expect the vendors to manage complete testing and provide them complete solutions. Hence it is important that the testing activities are well managed in terms of planning, executing and providing test results with complete visibility to the customer. Requests for testing come from different teams and locations of customers. Some of the activities, which could improve customer satisfaction are: Providing status of request at any instance to customer, Publishing results at a place accessible to all, Suggesting
areas of improvement and Comparison of previous results with the new results. Any company that has to look at its progress and move up the value chain and improve continuously has to necessarily collect data, analyze them and adopt corrective & preventive actions wherever necessary. The objectives of measurement in testing are:

- To better understanding how much/what/how...
- To better understand its Quality
- To develop more accurate estimates for future
- To better understand the Efficacy (power and effectiveness) of the Testing Process
- The data collected during testing process can also used by customers for improving certain processes at their end

7.2.1 Basic Testing Metrics

The collection of testing metrics starts after the unit testing. Unit testing is too unpredictable and too unstructured to make much sense from a metrics standpoint. However, the following metrics can be captured from unit testing through the end of the testing process [57,59,60,63].

Total errors uncovered: This is strictly a count of each error uncovered. By itself, it does not have as much meaning as when it is combined with other information for analysis.

Total errors by week: This measure gives an overall sense for how the testing process is going. You would expect that a vast number of errors would get caught in the early stages of testing. As time goes by, the number of errors uncovered per week should decrease. You may never reach a point were you find no more errors, but the team cannot complete testing if the number of errors is not tapering down close to zero.
**Number of hours spent correcting errors per week:** This metric provides a sense for the relative complexity of errors caught in testing. Early in the testing process, you may encounter errors that are time consuming to correct. However, as the testing cycle proceeds, the time that you spend correcting errors should decrease.

**Error correction hours per error uncovered:** This metric is similar to the prior metric. As you get further and further into the testing process, you should find that the time required fixing the typical error is getting shorter and shorter. If you uncover late errors that are time consuming to correct, it is usually a sign that the errors trace back to problems in the design or business requirements phases.

**Number of completed test cases/total number of test cases:** This metric gives the team a sense for how much work has been completed and how much is remaining.

**Budget vs. actual effort, cost, and duration:** These are standard metrics that point out how the project team is executing the testing process vs. the original estimates.

**Total testing cost/effort vs. total development cost/effort:** This metric provides some perspective on how one project spends its development time vs. other projects. If the average project spends 25% of its time in testing and your project spent 35% in testing, it might point out certain inefficiencies.
7.2.2 Slicing and Dicing the Errors

Error tracking can be tracked in a number of ways. They are all ways to provide further information to help in making decisions on how the testing process is going and where to focus the team’s testing time.

Errors by cause: The project team can track and categorize errors that are similar in nature. For instance, program errors will probably account for most errors in the beginning, but you may find later that errors can be traced to poor documentation or operator errors. Errors that are coming from a common cause can be attacked more vigorously.

Errors by subsystem: If a small number of subsystems have an unusually high number of errors, it can be a sign that more focus needs to be placed in those particular areas.

7.2.3 Production Defect Metrics

Problems uncovered in the testing process are referred to as errors. Once the application goes into production status, these same problems are referred to as defects.

Number of defects in production: This is an important metric to determine the overall effectiveness of the testing process. In a perfect world, an application would have no errors when it is implemented in production. However, there are always some. Counting these errors gives a sense of the stability of the system. You can capture similar metrics after implementation as you did during the testing process. For instance, you can capture the number of errors per week, which should decline over time, and the number of hours to fix production errors, which should also decline over time.
**Production downtime:** These metric measures the amount of time an application is down because of errors. This can show the business pain caused by production defects.

**Cost of defects:** These metric measures the cost of defects uncovered in production. This can include the cost of fixing the defects and the cost of people and resources that are idled or underutilized because of the downtime.

### 7.2.4 Quality and Productivity Metrics for Testing Teams

For a typical development project, the quality of work is considered superior when the customer is unable to find any bugs, code is easy to maintain and performs to meet his expectations. However the same is not applicable for project team, which is doing only testing activity. The testing team’s performance will be considered imperfect when there are

- Delivered Defects – Bugs identified by the customer after test Team’s testing
- Bugs identified during testing by the test team, which were not real bugs.
- Some of the other parameters that affect customer satisfaction are time taken for testing and effective utilization of hardware and tools etc.

**For Manual Testing**

\[
\text{Quality} = \frac{\text{Problems reported by customer}}{\text{Total Number of Test cases executed}}
\]  
(7.1)

\[
\text{Productivity} = \frac{\text{Total number of test cases executed}}{\text{Total Effort spent on testing (Man Hours)}}
\]  
(7.2)
For Test Automation

Quality = Problems reported in test scripts by customer / Total number of Test cases automated

Productivity = Total number of test cases automated / Total effort spent on Test Automation (Man Hours)

7.2.5 Efficiency of Test Logic

This is a measure of efficiency of each test and the selected set of tests. This does not include test execution efficiency (ex: automation) or efficient use of testing resources (infrastructure & human). It is understood that exhaustive testing is not a smart solution. Cost–effort-time constraints may force us to settle down to non-exhaustive testing. The challenge then is to pick the “right or best subset of tests” that will unearth maximum number of defects. Efficiency of test logic can be measured in some of the following ways:

- Average code coverage per test case – (the other side of this is sharpness/focus of the test case)
- Average state coverage per test case (for state based systems)
- Test thoroughness metric – fault exposure capability of a test case (mutation tests measure this)
- Minimum number of tests required for a given code coverage
- Minimum number of tests required for a given fault exposure
7.2.6 How Much to Test or When to Stop Testing?

This is a very important decision & a hard one to be made upfront, before we start the exercise. The basis of this can be one or more of the following:

- Testing till a time/effort/cost budget is exhausted – Testing stops when the budget is exhausted, irrespective of the defects detected or tests executed.

- Executing a pre-defined set of test cases – This is what happens most often and the selected number or set of tests do not usually have a very logical basis. This is an area with significant improvement opportunity. This has been discussed in more detail under “test efficiency”.

- Testing till average failure rate drops below a threshold – This is followed in case of mission critical systems. No matter what the cost/time/effort could be, testing must continue till the “max allowed failure rate” quality goal is reached.

- Achieving a set level of coverage - Test coverage has different meanings & measurement units in the context of different kinds of testing (Ex: % KLOC, % function points, % numbered requirements) – Coverage is quantitatively measured & testing must continue till the coverage criterion is met.

- Achieving a pre-defined residual defect density - Estimating upfront, the total number of expected defects & testing till a predefined % of these defects are
detected (& removed so that not more than a pre-define % of defects are left behind in the system)

7.3 Analysis of Test Results

After results of individual tests are gathered over a period of time, these test results should be subjected to detailed level analysis. This analysis should be statistically rigorous & should highlight trends/patterns over periods of observation [60,64,120,121,122]. The data collected by the testing projects can be used for various purposes like

- To measure the Effectiveness of the testing activity
- Trend analysis of test results over a release/ over multiple releases
- Determination of Achieved Test Goals & Quality Goals
- Earned Value Analysis
- Quality and Productivity improvement
- Identifying and improving the weak areas in the testing process
- Comparison between different testing groups (including customer’s testing group)
- Customer delight by managing testing better
- Estimation for future projects

7.3.1 Effectiveness of the Testing Activity

The defects in testing process can be:

- Delivered Defects – Defects identified by the customer after Team’s testing
- Bugs identified during testing by the team, are not true bugs
Data collected during the testing process, could be analyzed and effectively used to improve the testing effectiveness. Figure 7.1 is an illustration of a testing project’s defects data collection, and how it was used for improving the quality.

<table>
<thead>
<tr>
<th>Defects for Testing Team</th>
<th>%</th>
<th>Cum %</th>
<th>No of Defects</th>
</tr>
</thead>
<tbody>
<tr>
<td>Duplicate Defect</td>
<td>43</td>
<td>43.10</td>
<td>100</td>
</tr>
<tr>
<td>Functionality Not Clear</td>
<td>34</td>
<td>77.38</td>
<td>80</td>
</tr>
<tr>
<td>Not a defect</td>
<td>15</td>
<td>92.67</td>
<td>35</td>
</tr>
<tr>
<td>Configuration data Not properly Setup</td>
<td>6.90</td>
<td>99.57</td>
<td>16</td>
</tr>
<tr>
<td>Insufficient Information</td>
<td>0.43</td>
<td>100.00</td>
<td>1</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td><strong>232</strong></td>
<td></td>
</tr>
</tbody>
</table>

**FIGURE 7.1: PROJECT DEFECT DATA**

Causal Analysis was done on the defects, root causes identified and appropriate actions were initiated to avoid the repetition of the defects.

**FIGURE 7.2: PROJECT DEFECT DATA PARETO ANALYSIS**

A checklist was developed as the part of Defect Prevention for reporting the bugs detected during testing, thereby reducing the percentage of wrong bugs reported. Over a period of time the quality of testing process improved. The chart in Figure 7.3 shows the improvement trend observed in the project later.
7.3.2 Productivity Trend Analysis

Metrics analysis will also help a testing team, to see how they have performed over a period. A testing project analyzing its metrics will be able to detect any deteriorating trends and take suitable corrective actions immediately. The results of testing should be continuously evaluated, compared against past results, other project results and compared against industry benchmarks. As an example this testing project found that teams productivity trend was deteriorating during Mar and April '03 as shown in Figure 7.4. Immediately the analysis was done and corrective actions taken to recover from negative trend.

At this juncture the project team looked back and analyzed the reasons for the negative productivity trend and arrived at the following:

- Requests were coming from different customer teams
• Testing personnel were spending too much effort in communicating and providing the data to different persons/groups

• There was no single location where all the information was visible.

The project manager reviewed with the team and developed a tool where customer could log in his request and tester would take up the work from the tool. After testing, the tester would provide link to the test results in the tool itself. This way the project could save lot of time/effort, which resulted in complete satisfaction of the customer. Later, they also achieved a customer satisfaction rating of 5 on 5 for two consecutive periods. The productivity figures after introduction of the tool is shown in Figure 7.5.

7.3.3 Determination of Achieved Test Goals & Quality Goals

Quantitative measurement of residual defect density, failure rate, MTBF, RPN, availability etc., at release, can give a definitive picture of testing effectiveness. This data can be used for comparison against competition & can help in arriving at benchmarks.

![Productivity Trend](image)

FIGURE 7.5: PROJECT POSITIVE PRODUCTIVITY TREND

7.3.4 Earned Value Analysis

Actual cost of test performed and deviation from estimated cost should be formally determined at product release. This data should be used as the input for the next
release/or similar project initiation. This is essential for future testing project cost planning & control.

7.3.5 Identifying & Improving Weak Areas in the Testing Process
Defects analysis also helps in identifying the weak areas of the testing process, and will help us in strengthening those areas. As an example, a testing project found that 74% of defects in the automation process were injected during scripting and test planning stages. These were identified as weak areas and suitable corrective actions were taken to strengthen them.

![Defect Injected Stage (Test Automation)](image)

FIGURE 7.6: PROJECT DEFECT INJECTED STAGE

7.3.6 Comparison between Different Testing Groups
Data analysis helps in comparing the performance of different testing groups. As an example, testing projects executed by one of the vendor used metrics to compare with the customer’s testing group, thereby showcasing their value adds, to the customers.
7.3.7 Other Analysis

There can be many other analyses performed on the data collected to arrive at the decision of when to stop testing or how much testing is enough? The defect arrival rate over a period of time can help in arriving at these decisions.
7.3.8 Estimation for Future Projects

Estimation is one of the weak areas, which most software companies face. It is more so when a project is doing only testing activities. Unless different activities are closely monitored and measured, it is impossible to estimate for future/new activities. Measuring effort for different activities of testing will help in estimating for future activities. The organization can come up with the productivity figures over a period of time, which would aid in estimating future projects with greater degree of accuracy.

7.3.9 Organization Capability Baseline

The data collected for projects is used to arrive at Organization capability baseline. Testing Process performance can be captured in terms of the distributions of productivity, quality, schedule, effort and defects. This will allow project performance to be predicted using this baseline. The baseline should also define allowable or tolerance limits for performance variation on a specific project. Project Managers can use these limits to monitor the performance of the project processes.

7.4 Summary

The best way to manage a testing project is to manage the same objectively and not based on perception. The only way to really know how your testing is going is to measure it. Here is what you should keep in mind as you keep track of your testing.

- Testing metrics should be captured to provide a sense for how the entire testing process is progressing.
- The most important component of testing metrics is the capturing of the number of errors. In addition, further information can be gathered as to the cost and effort
associated with correcting the errors. With this information, valuable insight can be gathered into whether the testing process is under control or not.

Much of the same information that is captured during the testing process can also be gathered after the application goes into production. At that point, problems should continue to be tracked as defects. Performance of software testing process should be assessed with use of appropriate metrics. Improvement in metrics over period of time should be targeted. Unless testing process improvement is accomplished, the testers continue to make same errors and perform testing inefficiently time after time. The results of testing should be continuously evaluated, compared against past results, other project results and compared against industry benchmarks. Companies should look at adopting testing best practices and benchmarks and subject the processes to continuous improvement.