Notable Metrics in Software Testing

S. M. K. Quadri¹ and Sheikh Umar Farooq²
¹²P. G. Department of Computer Sciences, University of Kashmir
¹quadrismk@hotmail.com and ²shiekh.umar.farooq@gmail.com

ABSTRACT
Measurement is essential to achieving the basic management objectives of prediction, progress, and process improvement. We cannot build quality software, or improve our process, without measurement. Measurement activities keep us actively involved in, and informed of, product and process. Measurement in testing projects is a challenging, but indispensable element of a software testing process. Software testing today has a multitude of metrics for measuring different aspects of testing. Test metrics are an important indicator of the effectiveness of a software testing process. In this paper we present the various metrics that should be used in software testing. These test metrics may become an important indicator of the effectiveness and efficiency of a software testing process and may also identify risky areas that may need more testing.

KEYWORDS

1. INTRODUCTION
When you can measure what you are speaking about, and can express it in numbers, you know something about it; but when you cannot measure it, when you cannot express it in numbers, your knowledge is of meager and unsatisfactory kind; it may be the beginning of knowledge, but you have scarcely in your thoughts advanced to the stage of science". Hence, it is clear that everything should be measurable. If it is not measurable, we should make an effort to make it measurable [1]. The area of measurement is very important in every field, and we have mature and established metrics to quantify various things. However, in software engineering this “area of measurement” is still in its developing stage and may require significant effort to make it mature, scientific and effective. In applying measurements to software engineering, several types of metrics are available. Schulmeyer defines a metric as “A quantitative measure of the degree to which a systems, component or process possesses a given attribute” [2]. Software metrics are used to evaluate the software development process and the quality of the resulting product [3]. Software metrics aid evaluation of the testing process and the software product by providing objective criterion and measurements for management decision making. Their association with early detections and correction of problems make them important in software. Software test metrics have found good number of applications during testing. One area is the reliability estimation, where popular models are Musa’s basic execution time model and the Logarithmic Poisson execution time model. Source code coverage metrics are available that calculate the percentage of source code covered during testing. Test suite effectiveness may also be measured. Number of failures experienced per unit of time, number of paths, number of independent paths, number of “define-use” paths, percentage of statement coverage, percentage of branch conditions covered are also useful software metrics. They provide meaningful and timely information, which may help us to take corrective actions as and when required. Effective implementation of metrics may improve the quality of the software and may help us to deliver the software in time and within budget. Figure 1 shows the necessary measurements in testing which contribute to quality software.

2. SOFTWARE METRICS.
A metric is a quantifiable measurement of software product, process, or project that is directly observed, calculated, or predicted [4]. Metrics facilitate the quantification of some particular characteristic. Essentially, software metrics deals with the measurement of the software product and the process by which it is developed. They are quantifiable indices used to compare software products, processes, or projects or to predict their outcomes. Figure 2 shows the process of measurement. With Software metrics, we can:
• Monitor requirements,
• Predict development resources,
• Track development progress, and
• Understand maintenance costs.

3. CATEGORIES OF METRICS
The term used to describe a measurement of a particular attribute of a software project is a Software Metric. The Software Metrics that the QA team produces are concerned with the test activities that are part of the Test Phase and so are
formally known as Software Testing Metrics [5]. There are two broad categories of software metrics, namely product metrics and process metrics. Figure 3 shows the categories of Software Metrics.

![Image](https://via.placeholder.com/150)

**Figure 2:** Measurement Process.

**Figure 3:** Software Metrics Breakdown

Test product metrics measures provide information about the test state of the product and are generated by test execution and code fixes or deferment. Using these metrics we can measure the products test state and indicative level of quality, useful for product release decisions. Product metrics describe the characteristics of the product such as size, complexity, design features, performance, efficiency, reliability, portability, etc [6]. Test process metrics measures provide information about preparation for testing, test execution and test progress. They don’t provide information about the test state of the product and are primarily of use in measuring progress of the Test Phase. Process metrics describe the effectiveness and quality of the processes that produce the software product. Examples are effort required in the process, time to produce the product, effectiveness of defect removal during development, number of defects found during testing, maturity of the process [6].

### 3.1 PRODUCT METRICS FOR TESTING

These metrics provide information about the testing status of a software product. The data for such metrics are also generated during testing and may help us to know the quality of the product. Some of the basic metrics are given as:

1) Number of failures experienced in a time interval
2) Time interval between failures
3) Mean time between failures (MTBF).

4) Cumulative failures experienced up to a specified time.
5) Time of failure.
6) Estimated time for testing.
7) Actual testing time.

With these basic metrics, we may find some additional metrics as given below:

1) \( \% \) of time spent = (Actual time spent / Estimated testing time) \* 100
2) Average time interval between failures
3) Maximum and minimum failures experienced in any time interval
4) Average number of failures experienced in time intervals
5) Time remaining to complete the testing

We may design similar metrics to find the indications about the quality of the product.

### 3.2 PROCESS METRICS FOR TESTING

These metrics are developed to monitor the progress of testing, status of design and development of test cases and outcome of test cases after execution. Some of the basic process metrics are given below:

1) Number of test cases designed
2) Number of test cases executed
3) Number of test cases passed
4) Number of test cases failed
5) Test case execution time
6) Total execution time
7) Time spent for the development of a test case
8) Total time spent for the development of all test cases

On the basis of the above direct measures, we may design the following additional metrics, which may convert the base metric data into more useful information.

1) \( \% \) of test cases executed
2) \( \% \) of test cases passed
3) \( \% \) of test cases failed
4) Total actual execution time / total estimated execution time
5) Average execution time of a test case.

These metrics, although simple, may help us to know the progress of testing and may provide meaningful information to the testers and project manager. An effective test plan may force us to capture data and convert it into useful metrics both for process and product. This document also guides the organization for future projects and may also suggest changes in the existing processes in order to produce a good-quality maintainable software product. The description and example of each test metrics category is listed in table 1.

### 4. SOFTWARE TESTING METRICS – NEED AND IMPORTANCE

Software metrics are applicable to the whole development life cycle from initiation, when cost must be estimated to monitoring the reliability of the end product in the field, and the may that product changes over time with enhancement.
Software metrics hold importance in testing phase, as software testing metrics acts as indicators of software quality and fault proneness. Testing metrics exhibit trends and characteristics over time that would be indicative of the stability of the process [7]. The essential step is establishing test metrics is to identify the key software testing processes that can be objectively measured. Measuring software development and testing projects is a challenging, but essential component of a professional organization. Software project may be running over time and over budget and still have a high number of defects. Or, it may be on time and on budget and have an even higher number of defects. Measuring allows you to quantify your schedule, development, and testing efforts. When you measure your current project performance, you become better equipped to schedule and budget for future projects. A major percentage of software projects suffer from quality problems, which in turn requires new testing metrics to measure test processes effectively. Test metrics are key “facts” that project managers can use:

1) To understand their current position
2) To prioritize their activities to reduce the risk of schedule over-runs on software releases.

Test metrics are powerful risk management tool, help us to measure current performance. Importance of having test metrics [8]:

1) Provides a basis for estimation and facilitates planning for closure of the performance gap.
2) Provides a means for control/status reporting.
3) Identify risk areas that require more testing.
4) Quickly identifies and helps to resolve potential problems and identifies areas of improvement.
5) Test metrics provide an objective measure of the effectiveness and efficiency of testing.

### Table 1: Classification of metrics

<table>
<thead>
<tr>
<th>Category</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product Metrics</td>
<td>Describe the characteristics of the product</td>
<td>Size, Performance, Efficiency etc.</td>
</tr>
<tr>
<td>Process Metrics</td>
<td>Describe the effectiveness and quality of the process that produce the software product</td>
<td>Effort, Time, No. of defects found during testing</td>
</tr>
</tbody>
</table>

5. NOTABLE SOFTWARE TESTING METRICS

Testing metrics may help us to measure the current performance of any project. The collected data may become historical data for future projects. This data is very important because in the absence of historical data, all estimates are just the guesses. Hence, it is essential to record the key information about the current projects. Test metrics may become an important indicator of the effectiveness and efficiency of a software testing process and may also identify risky areas that may need more testing.

5.1 TIME

We may measure many things during testing with respect to time and some of them are given as:

1) Time required to run a test case
2) Total time required to run a test suite
3) Time available for testing
4) Time interval between failures
5) Cumulative failures experienced up to a given time
6) Time of failure
7) Failures experienced in a time interval

A test case requires some time for its execution. A measurement of this time may help to estimate the total time required to execute a test suite. This is the simplest metric and may help to estimate the testing effort. We may calculate the time available for testing at any point in time during testing, if we know the total time assigned for testing. Generally, the unit of time is seconds, minutes or hours, per test case. Total testing time may be defined in terms of hours. Time needed to execute a planned test suite may also be defined in terms of hours.

The failure pattern may help us to define the following:

1) Time taken to experience *n* failures
2) Number of failures in a particular time interval
3) Total number of failures experienced after a specified time
4) Maximum / minimum number of failures experienced in any regular time interval.

5.2 TEST EFFICIENCY (TE)

This metric determine the efficiency of the testing team in identifying the defects.

It also indicated the defects missed out during testing phase which migrated to the next phase.

\[
\text{Test Efficiency} = \frac{DT}{DT + DU} \times 100
\]

Where,

\( DT \) = Number of valid defects identified during testing.
\( DU \) = Number of valid defects identified by user after release of application. In other words, post-testing defect

The higher the value of this metric, the better is the review efficiency.

5.3 SOURCE CODE COVERAGE

We may like to execute every statement of a program at least once before its release to the customer. Hence, the percentage of source code coverage may be calculated as:

\[
\% \text{ of source code coverage} = \left( \frac{\text{Number of statements of a source code covered by test suite}}{\text{Number of statements of a source code}} \right) \times 100
\]

Higher values of this metric give confidence about the effectiveness of a test suite. We should write additional test cases to cover the uncovered portions of the source code.

Copy Right © INDIACom-2011 ISSN 0973-7529 ISBN 978-93-80544-00-7
5.4 QUALITY OF SOURCE CODE
We may know the quality of the delivered source code after a reasonable time of release using the following formula:

\[ \text{QSC} = \frac{\text{WDB} + \text{WDA}}{\text{S}} \]

Where
- WDB: Number of weighted defects found before release
- WDA: Number of weighted defects found after release
- S: Size of the source code in terms of KLOC.

The weight for each defect is defined on the basis of defect severity and removal cost. A severity is assigned to each defect by testers based on how important or serious the defect is. Lower values of this metric indicate fewer errors detected or the detection of less serious errors. We may also calculate the number of defects per execution test case. This may also be used as an indicator of source code quality, as the source code progressed through the series of test activities.

5.5 TEST CASE DEFECT DENSITY
This metric may help us to know the efficiency and effectiveness of our test cases.

Test case defect density = \( \frac{\text{Number of failed tests}}{\text{Number of executed test cases}} \) * 100

Where Failed test case: A test case that, when executed, produced an undesired output. Passed test case: A test case that, when executed, produced a desired output.

Higher values of this metric indicate that the tests are effective and efficient, because they are able to detect more number of defects.

5.6 DEFECT ACCEPTANCE
This metric determines the number of valid defects that testing team has identified during execution.

Defect Acceptance = \( \frac{\text{Number of Valid Defects}}{\text{Total Number of Defects}} \) * 100

The value of this metric can be compared with previous release for getting better picture.

5.7 BAD FIX DEFECT
Defect whose resolution give rise to new defect(s) are bad fix defect. This metric determines the effectiveness of defect resolution process.

Bad Fix Defect = \( \frac{\text{Number of Bad Fix Defect(s)}}{\text{Total Number of Valid Defects}} \) * 100

This metric gives the percentage of the bad defect resolution which needs to be controlled.

6. CONCLUSION
Software testing metrics are used to measure specific attributes of software product or processes during testing. Metrics alone will not improve testing, but they provide information that will help to focus and evaluate your product and process improvements, but it is very important to know which metrics to use and which not to. Different testing schools have different view about the usefulness and applications of software metrics. However, every school of thought accepts that “we cannot improve what we cannot measure; and we cannot control what we cannot measure”. In order to control and improve various activities, we should have “something” to measure such activities. This “something” differs from one school of thought to another.

REFERENCES