ABSTRACT

Suppose, we have developed software. Now we have to check its quality and functionality. Where does this newly developed software stands among previously released software of similar functionality? Is it better or worse? How can we further improve its quality and performance? How do we find out answer to all above questions? This is the point where we make use of metrics. Software metrics measures a software providing information about quality, completeness, performance to the developer. This information can further help the organization/developer to further improve their software in many aspects. In this paper, testing metrics are being studied in the different views. The testing metrics can be classified w.r.t. the process, project and product testing metric and in the other way manual, automation, common and performance testing metrics. To see what classification includes many types of testing metrics, we can combined different types and see to its effects.

Key words: Software Testing Metrics, Tool Command Language, Test case productivity, Requirement stability index, Test execution, Performance test efficiency.


1. INTRODUCTION

Metricis defined as a unit of measurement that computes results. Metric used for evaluation and quantification of a software are called as software metrics. This information is further used for improvisation of processes, products and services. Software metric characterizes a software system and places them at a specific position on a software quantitative scale.

Now a day, Software measurement has become an essential and crucial part of software development. With the help of software measurement, we can focus on specific characteristics of software and improvise them. However, further involvement of software measurement is still required as far as quality evaluation is concerned [4, 5, 6, 7]. Most of the
use of software metrics focuses mainly on cost assessment. However, we all know there are a number of aspects of a software which needs to be measured and having them measured can provide us a great deal of improvisation.

The essential qualities of a software metrics

- Easy to understand
- Precise and complete data
- Selection of metrics based on importance of stakeholders
- Data corresponding to either amendment or decision making
- Independent, Robust, reliable and valid Consistent and used over time

2. SOFTWARE METRICS LIFE CYCLE

Various steps involved in life cycle of software metric are as following:

- Categorization and prioritization of metrics.
- Classification of project specific metrics
- Collection of data for processing of metrics
- Communication with stakeholder/customers
- Capturing and verification of data
- Analyzing and dispensation of data
- reporting

![Analytical Lifecycle of metric](image)

**Figure 1** Analytical Lifecycle of metric

3. SOFTWARE TESTING

Software testing is a process to make sure of correctness of software i.e. the software developed is up to the standards set at the time of planning and is able to uphold expectations of the customers[2, 3].

There are basically two categories of software testing:

**Manual testing:** Test cases are prepared and executed manually by concerned testing engineer[1, 11]. It is anlaborious work that requires patience, skill, creativeness, innovative approach and open mindedness.

http://www.iaeme.com/IJCET/index.asp 109 editor@iaeme.com
Major problems of manual testing are:

- very time consuming,
- requirement of high investment in HR,
- low reliability,
- non-programmable and error prone

**Figure 2** Types of Software Testing

**Automation testing:** Programming languages such as JavaScript, Python or TCL (Tool Command Language) are used for preparation and execution of test cases. Automation can reduce human effort and cost to a great extent [9].

Major benefits of automation testing are:

- cost effectiveness, reusability and repeatable
- use of programming languages
- easy to understand, more reliability

4. TYPES OF SOFTWARE METRICS

Software metrics are classified into many categories. One method is to classify them in Manual testing metrics, Performance testing metrics, Automating testing metrics, and Common testing metrics. In another type of classification, they are categorised as project, process and product metrics.

4.1. Testing Metrics Classification I

There are mainly four types of software testing:

- Manual testing metrics
- Performance testing metrics
- Automating testing metrics
- Common testing metrics
4.1.1. Manual Testing Metrics

Test case productivity Metrics (TCP): It provides information about test progression. It represents number of test cases designed per hour.

Test case productivity = Total No of test steps/efforts in hours

For example:

<table>
<thead>
<tr>
<th>Test case name</th>
<th>No of steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>44</td>
</tr>
<tr>
<td>A2</td>
<td>39</td>
</tr>
<tr>
<td>A3</td>
<td>26</td>
</tr>
<tr>
<td>A4</td>
<td>37</td>
</tr>
<tr>
<td>A5</td>
<td>24</td>
</tr>
<tr>
<td>Total no</td>
<td>170</td>
</tr>
</tbody>
</table>

Total effort taken = 9 hours

TCP = 170/9 = 18.88 steps/ hours

Test case productivity of previous releases can be compared graphically for better understanding. An example is shown below in figure 4.
Test execution summary metrics: It represents the status of test cases whether failed, succeeded or not run. Figure 5 show an example of a pie chart showing statistical summary of test execution result of test cases of software.

Defect Acceptance Summary: It concludes total no of valid test whose identification is done under execution process.

Defect acceptance = \[\frac{\text{no of valid defects}}{\text{total no of defect}}\] * 100

Table 2 Defect Acceptance of Various Releases

<table>
<thead>
<tr>
<th>Releases</th>
<th>Valid defect</th>
<th>Total defects</th>
<th>Defect acceptance (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>44</td>
<td>92</td>
<td>48</td>
</tr>
<tr>
<td>II</td>
<td>65</td>
<td>102</td>
<td>64</td>
</tr>
<tr>
<td>III</td>
<td>67</td>
<td>123</td>
<td>54</td>
</tr>
<tr>
<td>IV</td>
<td>60</td>
<td>87</td>
<td>68</td>
</tr>
</tbody>
</table>

Like Test case productivity, defect acceptance of previous releases can also be compared graphically for better understanding. An example is shown below in figure 6.
Defect Rejection: It concludes total no of defect rejected during execution process. 
Defect rejection = [no of rejected defects/total no of defect] * 100

Table 3 Defect Rejection of Various Releases

<table>
<thead>
<tr>
<th>Releases</th>
<th>Invalid defect</th>
<th>Total defects</th>
<th>Defect rejection (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>13</td>
<td>80</td>
<td>16</td>
</tr>
<tr>
<td>II</td>
<td>14</td>
<td>83</td>
<td>17</td>
</tr>
<tr>
<td>III</td>
<td>17</td>
<td>128</td>
<td>13</td>
</tr>
<tr>
<td>IV</td>
<td>21</td>
<td>100</td>
<td>21</td>
</tr>
</tbody>
</table>

Figure 7 Defect Rejection Trend

A comparison of defect rejection percentage of previous release is shown in figure 7.

Bad fix Defect Metrics: Defects identified in testing process are resolved by the testing engineer. If by any chance, concerned defect is fixed but resolution is done in such a wrongly manner that it gives rise to a new defect occurrence, and then it is called bad fix defect metric. So this metric concludes effectiveness of defect resolution process.

Bad fix defect = [no of bad fix defects/total no of valid defects] / 100

Test execution Productivity metrics: It determines average no of tests executed per day. It may be defined as executions per day.

Test execution productivity = [Total no of test cases executed (Te)/ execution effort in hours] * 100

Where
Te = BTC + [T(1)*1] + [T(0.33)*0.33] + [T(0.66)*0.66]
BTC = no of test cases executed at least once
T(1) = no of test cases retested with 71% to 100% of total test case steps
T(0.66) = no of test cases retested with 41% to 70% of total test case steps
T(0.33) = no of test cases retested with 1% to 40% of total test case steps
Table 4 Test execution productivity

<table>
<thead>
<tr>
<th>TC</th>
<th>Base run effort</th>
<th>Rerun 1 status</th>
<th>Rerun 1 effort (1hr)</th>
<th>Rerun 2 status</th>
<th>Rerun 2 effort (1hr)</th>
<th>Rerun 3 status</th>
<th>Rerun 3 effort (1hr)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>1.3</td>
<td>T(0.66)</td>
<td>1</td>
<td>T(0.33)</td>
<td>0.52</td>
<td>T(1)</td>
<td>2</td>
</tr>
<tr>
<td>A2</td>
<td>1.6</td>
<td>T(0.33)</td>
<td>0.4</td>
<td>T(1)</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A3</td>
<td>2</td>
<td>T(1)</td>
<td>1.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A4</td>
<td>2.3</td>
<td>T(1)</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A5</td>
<td>2.7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 5 Types of Test Case and Values

<table>
<thead>
<tr>
<th>Types of Test Case</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base test case</td>
<td>5</td>
</tr>
<tr>
<td>T(1)</td>
<td>4</td>
</tr>
<tr>
<td>T(0.66)</td>
<td>1</td>
</tr>
<tr>
<td>T(0.33)</td>
<td>2</td>
</tr>
<tr>
<td>Total efforts (hr)</td>
<td>19.65</td>
</tr>
</tbody>
</table>

\[ Te = [5 + (1*4) + (1*0.66) + (2*0.33)] = 10.32 \]

**Test execution productivity** = \[ \frac{10.32}{19.65}*8 = 4.2 \text{ exe/day} \]

A comparison of previous releases has been shown in figure 8.

**Figure 8 Test Execution Productivity Trend**

**Test efficiency:** It determines how efficient the testing team to identify the defects is. It also may be defined as the no of defects missed out during testing process.

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

where

DT = no of valid defects identified during testing
DU = no of valid defects identified after release of software by the user.

Figure 9 shows an example test efficiency pattern of previous releases.
Impact of Software Testing Metrics on Software Measurement

**Figure 9** Test Efficiency Trend

**Defect Severity index (DSI):** It represents quality of the software i.e. under test and at the time of release of product. Release of product depends highly upon this metric.

**Figure 10** Defect Severity index Trend (all status)

Defect Severity index = \( \sum \) (Severity index*no of valid defects for this severity)/total no of valid defects

DSI can be categorised into two types:
- DSI for all status defects – It represents quality of product under testing process
- DSI for open status defects – It represents quality of product when released.

DSI (open) = \( \sum \) (Severity index*no of open valid defects for this severity)/total no of open valid defects.

**Figure 11** Defect Severity index Trend (for open status)
4.1.2. Performance Testing Metrics

**Performance scripting productivity**: It quantifies performance of the script written for testing. An example is shown in figure 12.

Performance scripting productivity = total no of operation performed/efforts in hours

<table>
<thead>
<tr>
<th>Operation performed</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. Of clicks</td>
<td>20</td>
</tr>
<tr>
<td>No of checking point</td>
<td>7</td>
</tr>
<tr>
<td>No of input parameters</td>
<td>8</td>
</tr>
<tr>
<td>No of relationship pattern</td>
<td>10</td>
</tr>
<tr>
<td>Total operations</td>
<td>45</td>
</tr>
</tbody>
</table>

Total effort for writing script = 9 hours

Performance scripting productivity = \( \frac{45}{9} = 5 \) operations /hour

![Performance Scripting Productivity](image)

**Figure 12** Product Scripting Productivity Trend

**Performance execution summary**: It represents the status of test cases whether failed, succeeded or not run.

**Performance execution data client side**: This metric observes client side keenly and provides data such as:

- No of active users
- No of transactions per minute
- Responding time
- Average spending time
- Errors per minute
- Entries per minute

**Performance execution data server side**: This metric observes client side keenly and provides data such as:

- CPU Usage
- Total connections per minute
- Data transfer per minute
- Memory usage
Performance test efficiency (PTE): It determines how efficient is the testing team to identify the defects. It also may be defined as the percentage of requirements met at release of product.

\[
PTE = \frac{(\text{Demands met during PT}) - (\text{Demands met at release})}{\text{demands met during PT}}
\]

Data required for quantification of this metric is as:

- Average no of transactions per minute
- Average Responding time
- Average spending time
- Errors per minute
- Entries per minute

For e.g. if

Demands met during PT = 5
Demands not met after signoff of PT = 1

\[
PTE = \frac{5}{5+1} \times 100 = \frac{5}{6} \times 100 = 83\%
\]

Performance severity index: It represents quality of the software i.e. under test and at the time of release of product. Decision of release of product depends highly upon this metric.

4.1.3. Automation Testing Metrics

Automation scripting productivity (ASP): It quantifies performance of the script written for automation. An example is shown in figure 8.

Performance scripting productivity = total no of operation performed/efforts in hours

- Examples of operations can be as follows:
- Number Of clicks
- Number of checking point
- Number of input parameters
- Number of relationship pattern

Figure 13 Performance Test Efficiency Trend
Table 7 Types of operations

<table>
<thead>
<tr>
<th>Operation performed</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. Of clicks</td>
<td>20</td>
</tr>
<tr>
<td>No of checking point</td>
<td>5</td>
</tr>
<tr>
<td>No of input parameters</td>
<td>5</td>
</tr>
<tr>
<td>No of relationship pattern</td>
<td>10</td>
</tr>
<tr>
<td>Total operations</td>
<td>40</td>
</tr>
</tbody>
</table>

Total effort for writing script = 10 hours

Performance scripting productivity = 40/10 = 4 operations /hour

![Figure 14 Automation scripting productivity trend](image)

**Automation Test Execution Productivity metrics:** It determines average no of automated tests executed per day. It may be defined as executions per day.

Test execution productivity = \[\text{Total no of test cases executed (ATE)/ execution effort in hours}] \times 100

Where

\[\text{ATE} = \text{BTC} + ([\text{T}(0.33)\times0.33] + [\text{T}(0.66)\times0.66 + \text{T}(1)\times1])\]

\[\text{BTC} = \text{no of test cases executed atleast once}\]

\[\text{T}(1) = \text{no of test cases retested with 71% to 100% of total test case steps}\]

\[\text{T}(0.66) = \text{no of test cases retested with 41% to 70% of total test case steps}\]

\[\text{T}(0.33) = \text{no of test cases retested with 1% to 40% of total test case steps}\]

**Automation coverage:** This metrics tell us about how many manual test cases have been automated using automation technique[13].

Automation coverage = \[\text{No of manual test cases automated/total no of manual test cases}\]

For example, if there are 500 total manual test cases and out of which 400 have been automated, then automation coverage will be \[400/500\] i.e. equal to 80%.

**Cost comparison metrics:** This metric provides an estimate of cost difference between manual testing and automation testing [14, 15].

Cost (Manual) = execution efforts in hours/ billing rate

Cost (auto) = tool purchase cost (single time investment) + maintenance cost + script development cost + efforts in hours/ billing rate
4.1.4. Common Testing Metrics

Effort variance (EV): This metrics tells us about percent deviation between estimated effort and actual effort.

\[ EV = \frac{(\text{actual effort} - \text{estimated effort})}{\text{estimated effort}} \times 100 \]

Table 8 Effort Variance Values

<table>
<thead>
<tr>
<th>Project name</th>
<th>Actual effort</th>
<th>Estimated effort</th>
<th>Effort variance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project 1</td>
<td>106</td>
<td>100</td>
<td>6</td>
</tr>
<tr>
<td>Project 2</td>
<td>79</td>
<td>85</td>
<td>-7.1</td>
</tr>
<tr>
<td>Project 3</td>
<td>100</td>
<td>106</td>
<td>-5.6</td>
</tr>
<tr>
<td>Project 4</td>
<td>200</td>
<td>191</td>
<td>4.5</td>
</tr>
</tbody>
</table>

Figure 15 Effort Variance trend

Schedule Variance [SV]: This metrics tells us about percent deviation between number of days estimated for completion of a process and actual time taken for the process completion.

\[ SV = \frac{\text{actual no of days} - \text{estimated no of days}}{\text{estimated no of days}} \times 100 \]

Table 9 Schedule variance values

<table>
<thead>
<tr>
<th>Project name</th>
<th>Actual time</th>
<th>Estimated time</th>
<th>Schedule</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project 1</td>
<td>119</td>
<td>111</td>
<td>+6.3</td>
</tr>
<tr>
<td>Project 2</td>
<td>153</td>
<td>165</td>
<td>-4.9</td>
</tr>
<tr>
<td>Project 3</td>
<td>100</td>
<td>95</td>
<td>+5</td>
</tr>
<tr>
<td>Project 4</td>
<td>200</td>
<td>215</td>
<td>-7.5</td>
</tr>
</tbody>
</table>

Figure 16 Schedule Variance trend
**Scope variance (SC):** This metrics tells us about percent deviation of stability of previous releases.

\[ SC = \left( \frac{\text{Total scope} - \text{Previous scope}}{\text{Previous scope}} \right) \times 100 \]

Where

Total scope = Previous scope + New Scope, if Scope inc
Total scope = Previous scope - New Scope, if Scope dec

**Defect aging:** This metric gives an indication of average time taken to fix an identified defect. Its unit can be either days or hours.

Defect aging = difference in time created and time resolved

**Defect fix retest:** It evaluates the efficiency of retesting previously fixed defects if they are working properly

**Current quality ratio:** This metrics tell us about efficient functionality of the process.

Current quality ratio = total no of test cases without any defects/total number of test procedures

**Error discovery rate:**

Error discovery rate = Total number of defects identified/ number of test processes executed

**Defect density:** This metrics is calculated in terms of line of codes of the software [12]. It can be defined as the no of defects identified per thousand lines of code (KLOC).

\[ \text{Defect density} = \frac{\text{Total number of defects}}{\text{KLOC}} \]

OR

Total number of defects/no of modules (components)

**Quality of fixes:** This metric evaluates the effects of software fixing on functionality of previously working processes whether function is adversely affected or no effect.

Quality of fixes = Total no of defects reopened/ total no of defects fixed

**Defect removal efficiency / defect gap analysis:** This Metric quantifies the ability of development team to remove an identified valid defect [13, 14] and preventing it from reaching to customer.

Defect removal efficiency = \[\left( \frac{\text{total number of valid defects resolved by testing team}}{\text{total defects identified during TC} - \text{total no of invalid defects}} \right) \times 100\]

For example in a testing cycle, 200 defects were observed by development experts, out of which 40 were found to be invalid defects. Expertise was able to resolve 120 defects out of remaining ones. So, Defect removal efficiency will be \[\frac{120}{(200 - 40)} \times 100 = 75\%\].

**Problem Reports:** It quantifies prioritized problems of the software i.e. some problems needs to be resolved quickly whereas some can be ignored for the time being.

### 4.2. Testing Metrics Classification II

In another way, metrics can also be categorised as project, process and product. These are

- Project Metrics
- Product Metrics
- Process Metrics
4.2.1. Project Metrics

Software project metrics are used to adapt project workflow and technical activities, to avoid development schedule delays, to mitigate potential risks, and to assess product quality on an on-going basis. These include Reliability, Availability, Usability; Customer reported defects, Productivity, Rate of Requirements Change, Estimation Accuracy, Cost, Scope, Risk etc. Metrics not defined before are described below:

Requirement stability index: This metric evaluates how change in magnitude and impact of requirements affects a project

\[
RSI = \left( \frac{\text{no of changed} + \text{no of deleted} + \text{no of added}}{\text{total no of initial requirements}} \right) \times 100
\]

Customer reported defects: These are defined as defect reports per customer-month

Schedule variance for a phase: It is defined as deviation between planned and actual schedule for the phases within a project.

\[
SV \text{ for a phase} = \frac{\text{actual calendar days for a phase} - \text{estimated calendar days for a phase}}{\text{estimated calendar days for a phase}} \times 100
\]

Effort variance for a phase: It is defined as deviation between planned and actual effort for the various phases within a project.

\[
EV \text{ for a phase} = \frac{\text{actual effort for a phase} - \text{estimated effort for a phase}}{\text{estimated effort for a phase}} \times 100
\]

- Schedule variance
- Size variance
- Productivity (for test case preparation)
- Productivity (for test case execution)
- Productivity (defect fixation)
- Productivity (defect detection)

4.2.2. Process Metrics

These metrics provide us an indication whether the process appears to be “working normally” or not. These metrics allow making changes while there is still a chance to have an impact on the project. These are useful during the development and maintenance process to identify problems and areas for improvement. These include Reliability growth pattern, Pattern of defects found (arrivals) during testing, responsiveness of fixing, fix quality etc. Metrics not defined above are described below:

Cost of quality: It is a measure in terms of money for quality performance in organization.

\[
\text{Cost of quality} = \frac{\text{review + testing = verification review + verification testin + QA + configuration management + management + training + rework review + rework testing}}{\text{total effort}} \times 100
\]

Cost of poor quality: It is defined as the cost of implementing imperfect processes and products.

\[
\text{Cost of poor quality} = \frac{\text{rework effort}}{\text{total effort}} \times 100
\]

Test cases Passed: This metric determines how many test cases are passed out of total executed test cases in percentage value.

\[
\text{Test cases Passed} = \left( \frac{\text{No. of Test cases Passed}}{\text{Total no. of Test cases Executed}} \right) \times 100
\]
**Test cases Failed:** This metric determines how many test case are failed out of total executed test cases in percentage value.

\[
\text{Test cases Failed} = \left( \frac{\text{No. of Test cases Failed}}{\text{Total no. of Test cases Executed}} \right) \times 100.
\]

**Test cases Blocked:** This metric determines how many test case are blocked out of total executed test cases in percentage value.

\[
\text{Test cases Blocked} = \left( \frac{\text{No. of Test cases Blocked}}{\text{Total no. of Test cases Executed}} \right) \times 100.
\]

- Test Efficiency
- Test Design Coverage
- Test Execution Productivity
- Test Case Preparation Productivity
- Review Efficiency
- Test Coverage productivity

### 4.2.3. Product Metrics

Software product metrics describes project team’s ability, to perform project execution example. These include Reliability, Availability, Usability, Customer reported defects, Productivity Etc. Metrics not defined before are described below:

**Defect leakage:** this is used to review the efficiency of testing process before UAT. Defect Leakage is the Metric which is used to identify the efficiency of the QA testing i.e., how many defects are missed / slipped during the QA testing.

\[
\text{Defect Leakage} = \left( \frac{\text{No. of Defects found in UAT}}{\text{No. of Defects found in QA testing}} \right) \times 100
\]

Suppose, During Development & QA testing, we have identified 100 defects. After the QA testing, during Alpha & Beta testing, end user / client identified 40 defects, which could have been identified during QA testing phase.

Review efficiency: It is defined as the efficiency in harnessing/ detecting review defects in the verification stage. Review efficiency = (number of defects caught in review)/ total number of defects caught) \times 100

**Residual defect density** = (total number of defects found by customer)/ (Total number of defects including customer found defects) \times 100.
5. TESTING METRICS IN THE VIEW OF PRODUCT, PROCESS AND PROJECT IN ADDITION OF THE MANUAL, AUTOMATION, PERFORMANCE AND COMMON TESTING METRICS

We have seen the classification of the testing metrics in two ways. In this step, we combined these two views. Firstly, we classify the considered metrics into project, process and product. Secondly, we identify the manual, automation, performance and manual metrics in each of the previous classification.

Table 10 Testing Metrics in the view of process, project, product in addition of common, manual, automation and performance

<table>
<thead>
<tr>
<th>Metric Type</th>
<th>Project Metrics</th>
<th>Process Metrics</th>
<th>Common Testing Metrics</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMMON TESTING METRICS</td>
<td>Schedule Variance</td>
<td>Size Variance</td>
<td>Requirement Stability Index</td>
</tr>
<tr>
<td></td>
<td>Effort Variance</td>
<td></td>
<td>Schedule Variance For A Phase</td>
</tr>
<tr>
<td></td>
<td>Size Variance</td>
<td></td>
<td>Effort Variance For A Phase</td>
</tr>
<tr>
<td>MANUAL TESTING METRICS</td>
<td>Productivity (For Test Case Preparation)</td>
<td></td>
<td>Productivity (Defect Fixation)</td>
</tr>
<tr>
<td></td>
<td>Productivity (For Test Case Execution)</td>
<td></td>
<td>Productivity (Defect Detection)</td>
</tr>
<tr>
<td>AUTOMATION TESTING METRICS</td>
<td>Cost of Poor Quality</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Cost of Quality</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MANUAL TESTING METRICS</td>
<td>Test Efficiency</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Test Design Coverage</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Test Execution Productivity</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Test Case Preparation Productivity</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PERFORMANCE TESTING METRICS</td>
<td>Review Efficiency</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Test Cases Passed</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Test Cases Failed</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Test Cases Blocked</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Test Coverage Productivity</td>
<td></td>
<td></td>
</tr>
<tr>
<td>COMMON TESTING METRICS</td>
<td>Residual Defect Density</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Defect Removal Effectiveness</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Quality of Fixes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PRODUCT METRICS</td>
<td>Error Discovery Rate</td>
<td></td>
<td></td>
</tr>
<tr>
<td>COMMON TESTING METRICS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Defect Density</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Defect Leakage</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Defect Removal Efficiency</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PERFORMANCE TESTING METRICS</td>
<td>Performance Execution Data Client Side</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Performance Execution Data Server Side</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In the table 10, we can see that the process metrics does have all four types metrics i.e. manual, automation, performance and common metrics. The percentage of said metrics is shown with the help of pie chart in the fig. 17.
The project metrics do have only manual and automation, testing metrics. The percentage of said metrics is shown with the help of pie chart in the fig. 18.

Figure 17 Distribution of various types of process metrics

Figure 18 Distribution of various types of project metrics

Figure 19 Distribution of various types of process metrics

Figure 20 Distribution of various types of project metrics
The product metrics does have common and performance testing metrics. The percentage of said metrics is shown with the help of pie chart in the fig. 21.

![Pie Chart](image)

**Figure 21** Distribution of various types of product metrics

6. CONCLUSIONS

The testing metrics is a foundation to the testing effort estimation. If we consider appropriate testing metrics to measure the effort, then it will reduce the chance to failure. The automation testing need to be preferred only if, it has chances to decrease the manual efforts. The process metrics include the all types of metric. But, it doesn’t mean that we ignored the impact of the project and product metrics. The balanced between all types of metrics only gives the best effect. We can explore testing metrics further to find the optimal balance between metrics.

REFERENCES


Monika, Pradeep Kumar Bhatia


http://www.iaeme.com/IJCET/index.asp

editor@iaeme.com