# Software Testing Tools Effectiveness for Quantification by Metrics

[1]Ajay Singh Dhabariya, [2]Shiwanne Rao

[1]Assistant Professor, [2]M.Tech. Scholar
Shrinathji Institute of Technology & Engineering,

***Abstract*: An automated testing tool supports the testers to calculate the quality of software by testing the software automatically. To calculate the quality of software there is always a requirement of good testing tools, which fulfil the testing requirement. There is a wide range of testing tools available in the market and they vary in approach, quality, usability and other characteristics. Selecting the correct testing tool for software there is a requirement of approach to prioritize them on the basis of characteristics. We propose a set of metrics for measuring the characteristics of the automated testing tools for examination and selection of automated testing tools. A new extended model which is proposed provides the metrics to calculate the effectiveness of functional testing tools on the basis of operability. The industry will be benefited as they can use these metrics to evaluate functional tools and they can further make selection of tool for their software required to be tested and hence reduce the testing effort, saving time and gaining maximum monetary benefit.**

***Keywords*: Software Testing, Software Metrics, Automated Testing Tools, Tool Evaluation**

## 1. Introduction

Testing is a tiresome part of the software development process. There are a lot of various automated software testing tools currently. Some of these tools are merely able to perform specific kinds of testing. Automated testing tools helps the testers to quantify the quality of software by testing the software. Although there is a wide range of testing tools available in the market and they vary in approach, quality, usability and other characteristics.

Selecting a right testing tool is very cumbersome. To select the appropriate testing tool there is a requirement of a way to prioritize them on the basis of characteristics. Evaluation of software testing tools effectiveness is becoming an important factor to be considered for software testing and assessment, especially for critical software. Metrics for evaluating automatic software testing tools were previously proposed by James B Michal,B B Synder and B J Bossuyt[1-5].

A new extended model which is proposed provides the metrics to calculate the effectiveness of any Black Box Testing tool on the basis of functionality and operability. This will help the students how to select the appropriate tools for the corresponding software. The industry will also be benefited, they can use the metrics to evaluate any tool and make the priority on the basis of analysis they can further make selection of tool for their software to be tested and reduce the testing effort hence saving time and gaining maximum monetary benefit.

## 2. Metrics

Quality measurement is usually expressed in terms of metrics. Software metric is a measurable property which is an indicator of one or more of the quality criteria that we are seeking to measure. As such, there are a number of conditions that a quality metric must meet. The history of software metrics began with counting the number of line of codes.[21] It was assumed that more line of codes implied more complex programs, which in turn were more likely to have errors.

### 2.1 Traditional Metrics

#### 2.1.1 Cyclomatic complexity

Cyclomatic complexity [1, 5] measures the amount of decision logic in a single software module. It is used for two related purposes in the structured testing methodology. First, it gives the number of recommended tests for software. Second, it is used during all phases of the software lifecycle, beginning with design, to keep software reliable, testable, and manageable. Cyclomatic complexity is based entirely on the structure of software's control flow graph. The cyclomatic number in graph theory is defined as:

CC = **e - n + 1**

The cyclomatic complexity definition for program control flow graphs is derived from the cyclomatic number formula by simply adding one to represent the contribution of the virtual edge. This definition makes the cyclomatic complexity equal the number of independent paths through the standard control flow graph model, and avoids explicit mention of the virtual edge.

$$M = V(G) = e - n + 2p$$

where V(G) is the cyclomatic number of G, e is the number of edges, n is the number of nodes, and p is the number of unconnected parts of G.

**2.1.2 Function point (FP)**

Function point [1,6,3] is a metric that may be applied independent of a specific programming language. To determine FP, an Unadjusted Function Point Count is calculated. UFC is found by counting the number of external inputs, external outputs, external inquiries, external files, and internal files. UFC is then calculated via:

**UFC = Σ1->15 (number of items of variety i) x (weight of i)**

Technical Complexity Factor is determined by analyzing fourteen contributing factors. The TCF is then found through the equation:

**TCF = 0.65 + 0.01Σ1->14 Fi**

**2.1.3 Halstead**

Halstead [13] created a metric founded on the number of operators and operands in a program. His software-science metric is based on the enumeration of distinct operators and operands as well as the total number of appearances of operators and operands. With these counts, a system of equations is used to assign values to program level, program difficulty, potential minimum volume of an algorithm, and other measurements.

A module's (M) fan-in is defined as the number of local flows that terminate at M plus the number of data structures from which information is retrieved by M. Fan-out is defined as the number of local flows that emanate from M plus the number of data structures that are updated by M. IFC is then found by summing the LOC of M and the square of the product of M's fan-in and fan-out. Shepperd removed LOC to achieve a metric more directly related to information flow

**IFC(M) = LOC(M) + [fan-in(M) x fan-out(M)]$^2$**

**2.2.1 Object Oriented Software metrics**

Object-oriented design and development has become popular in today's software development environment. An object-oriented program paradigm uses localization, encapsulation, information hiding, inheritance, object abstraction and polymorphism, and has different program structure than in procedural languages.

**2.2.1.1 Metric 1: Weighted Methods per Class (WMC)**

WMC is a sum of complexities of methods of a class. Consider a Class C1 with Methods M1…Mn that are defined in the class. Let c1…cn be the complexity of the methods1 [1, 6]. Then:

$$\mathrm{WMC} = \sum_{i=1}^{n} c_i$$

WMC measures size as well as the logical structure of the software. The number of methods and the complexity of the involved methods are predictors of how much time and effort is required to develop and maintain the class.

**2.2.1.2 Metric 2: Depth of Inheritance Tree (DIT)**

The depth of a class within the inheritance hierarchy is the maximum length from the class node to the root of the tree, measured by the number of ancestor classes. The deeper a class is in the hierarchy, the greater the number of methods it is likely to inherit, making it more complex to predict its behavior. Deeper trees constitute greater design complexity [1, 6], since more methods and classes are involved. The deeper a particular class is in the hierarchy, the greater potential reuse of inherited methods. For languages that allow multiple inheritances, the longest path is usually taken.

**2.2.1.3 Metric 3: Number of Children (NOC)**

Number of children metric equals to number of immediate subclasses subordinated to a class in the class hierarchy. Greater the number of children, greater the reuse, since inheritance is a form of reuse. Greater the number of children, the greater the likelihood of improper abstraction of the parent class [1,6,3]. If a class has a large number of children, it may be a case of misuse of sub classing. The number of children gives an idea of the potential influence a class has on the design. If a class has a large number of children, it may require more testing of the methods in that class.

#### 2.2.1.4 Metric 4: Coupling between object classes (CBO)

CBO for a class is a count of the number of other classes to which is coupled. CBO relates to the notion that an object is coupled to another object if one of them acts on the other [1,6]. Excessive coupling between object classes is detrimental to modular design and prevents reuse. The more independent a class is, the easier it is to reuse it in another application.

#### 2.2.1.5 Metric 5: Response For a Class (RFC)

The response set of a class is a set of methods that can potentially be executed in response to a message received by and object of that class2. RFC measures both external and internal communication, but specifically it includes methods called from outside the class, so it is also a measure of the potential communication between the class and other classes [1,6,14]. RFC is more sensitive measure of coupling than CBO since it considers methods instead of classes.

#### 2.2.1.6 Metric 6: Lack of Cohesion in Methods (LCOM)

The LCOM [1, 6] is a count of the number of method pairs whose similarity is 0 minus the count of method pairs whose similarity is not zero. The larger the number of similar methods, the more cohesive the class, which is consistent with traditional notions of cohesion that measure the inter-relatedness between portions of a program. If none of the methods of a class display any instance behavior, they have no similarity and the LCOM value for the class will be zero. Cohesiveness of methods within a class is desirable, since it promotes encapsulation. Lack of cohesion implies classes should probably be split into two or more subclasses.

## 3. Prior Work on Metrics

The Institute for Defense Analyses (IDA) published survey reports on tools for testing software [1]. Although the tool explanations contained in those reports are dated, the analyses provide a historical frame of reference for the recent advances in testing tools and identify a large number of measurements that may be used in assessing testing tools. For each tool, the report details different types of analysis conducted, the capabilities within those analysis categories, operating environment requirements, tool-interaction features, along with generic tool information such as price, graphical support, and the number of users.

The Software Technology Support Center (STSC) at Hill AFB works with Air Force software organizations to identify, evaluate and adopt technologies to improve product quality, increase production efficiency, and hone cost and schedule prediction ability. Section four of their report discusses several issues that should be addressed when evaluating testing tools and provides a sample tool-scoring matrix.

Brett Daniel, University of Illinois at Urbana-Champaign, Urbana, Illinois 61801, To give a broad overview of the process he propose to predict effectiveness of an automatic test generation tool. The process has two parts training a decision tree using code with known coverage characteristics and using the tree to predict coverage on new code. First, extract many metrics that characterize method structure. Second, run the automatic testing tool to produce a suite.

A thorough analysis was conducted by J. Thatcher. Evaluation and repair tools. His evaluation was aimed at determining the cost/benefit ratio and helping potential customers to select the most appropriate tool. In addition to considering costs, availability, and accuracy, the evaluation scope was quality of use in general. The method used is heavily based on manual and systematic inspection of the results produced by the tools on selected test pages, and is therefore less generally applicable than the method proposed in this paper, as it requires carefully prepared test files and a long and subjective analysis of the results.

In a more recent paper, Ivory and her colleagues aimed at evaluating quality of use of testing tools. They performed an experiment where web designers were asked to use testing tools and to modify web sites accordingly to what tools suggested. Then, in a second experiment, the authors determined how effective such changes were for disabled web site visitors.

Apart from the criteria defined by Poston and Sexton there is no founded approach for deriving evaluation criteria for test tools. Since the criteria proposed by Poston and Sexton focus on company specific criteria or on criteria requiring a high effort to be evaluated e.g. test effort or test quality, these criteria do not apply for a pre-selection of the test tools. The criteria mentioned by Poston and Sexton represent a subset of the criteria systematically derived in our approach.

Commercial test tool evaluations offered by OVUM and CAST, most of the non-commercial ones as proposed, appraise only a small subset of the test tools/frameworks available. These works concentrate on a detailed evaluation of products offered by the market leaders requiring extensive evaluation of the test tools. In contrast, in these thesis focus on more coarse-grained criteria

enabling an effective pre-selection for users. Furthermore, this report does not restrict the tools to be evaluated to a minimal set determined by the market leaders so that a wide range of test tools can be considered. The criteria defined in the CAST reports are not available.

## 4. Our Approach

The project suggests some of the metrics which can be used to discover the suitable automated software testing tool. These metrics are been derived on the functional and operational basis. The metrics are designed so they produced different values when applied to different testing tools. They can produce similar values also for different metrics and different testing tools. The suite of metrics to evaluate and select software testing tools carries the following properties: the metrics reveal smoothness in that they generate unlike values when applied to different testing tools. The metrics is finite in count and in very few cases they may provide similar values for few tools; usually they provide different values when applied to unlike testing tools.

### 4.1 Operational Metrics

#### 4.1.1 Maturity and Customer Base (MCB)

There are several providers of automated testing tools vying for the business of software testers. Tools that have achieved considerable maturity typically do so as a result of customer satisfaction in the tool's ability to adequately test their software. This satisfaction leads to referrals to other users of testing tools and an increase in the tool's customer base.

$$\mathbf{MCB = M + CB + P}$$

where M (maturity) is the number of years tool (and its previous versions) have been applied in real world applications, CB (customer base) is the number of customers who have more than one year of experience applying the tool, and P (projects) is the number of previous projects of similar size that used the tool Care must be taken in evaluating maturity to ensure the tool's current version does not depart too far from the vendor's previous successful path.

#### 4.1.2 Tool Management (TM)

As software projects become larger and more complex, large teams are used to design, encode, and test the software. Automated testing tools should provide for several users to access the information while ensuring proper management of the information. Possible methods may include automated generation of reports to inform other testers on outcome of current tests, and different levels of access.

$$\mathbf{TM = AL + ICM}$$

Where AL (access levels) is the number of different access levels to tool information, and ICM (information control methods) is the sum of the different methods of controlling tool and test information.

#### 4.1.3 Ease of Use (EU)

A testing tool must be easy to use to ensure timely, adequate, and continual integration into the software development process.

$$\mathbf{EU = LTFU + RFU + RCU + OTFU + OFCU + LTUEPV}$$

where LTFU is the learning time for first users, RFU is the retain ability of procedure knowledge for frequent users, RCU is the retain ability of procedure knowledge for casual users, OTFU is the average operational time for frequent users, OTCU is the average operational time for casual users and LTUEPV is the learning time of the users having working experience on previous versions.

#### 4.1.4 Test Case Generation (TCG)

The ability to automatically generate and readily modify test cases is desirable. Testing tools which can automatically generate test cases based on parsing the software under test are much more desirable that tools that require testers to generate their own test cases or provide significant input for tool generation of test cases. Availability of functions to create new test cases based on modification to automatically generated test cases greatly increases the tester's ability to observe program behavior under different operating conditions.

$$\mathbf{TCG = ATG + TRF}$$

#### 4.1.5 Tool Support (TS)

The level of tool support is important to ensure efficient implementation of the testing tool, but it is difficult to objectively measure. Technical support should be available to testers at all times testing is being conducted, including outside traditional weekday working hours. This is especially important for the extensive amount of testing frequently conducted just prior to product release.

$$\mathbf{TS = ART + ARTAH + ATSD – DI}$$

ART is the average response time during scheduled testing schedule, ARTAH is the average response time outside scheduled testing schedule, ATSD is the average time to search documentation for desired information, and DI is the documentation inadequacy measured as the number of unsuccessful searches of documentation.

#### 4.1.6 User Control (UC)

Automated testing tools that provide users expansive control over tool operations enable testers to effectively and efficiently test those portions of the program that are considered to have a higher level of criticality, have insufficient coverage, or meet other criteria determined by the tester. UC is defined as the summation of the different portions and combinations of portions that can be tested. A tool that tests only an entire executable program would receive a low UC value. Tools that permit the tester to identify which portions of the executable will be evaluated by tester-specified test scenarios would earn a higher UC value. Tools that will be implemented by testing teams conducting a significant amount of regression testing should have a high UC value to avoid retesting of unchanged portions of code.

#### 4.1.7 Estimated Return on Investment (EROI)

Vendors may also be able to provide similar statistics for their customers currently using their tools.

$$\textbf{EROI} = (\textbf{EPG} \times \textbf{ETT} \times \textbf{ACTH}) + \textbf{EII} - \textbf{ETIC} + (\textbf{EQC} \times \textbf{EHCS} \times \textbf{ACCS})$$

Where EPG is the Estimated Productivity Gain, ETT is the Estimated Testing Time without tool, ACTH is the Average Cost of One Testing Hour, EII is the Estimated Income Increase, ETIC is the Estimated Tool Implementation Cost, EQC is the Estimated Quality Gain, EHCS is the Estimated Hours of Customer Support per Project, and ACCS is the Average Cost of One Hour of Customer Support.

## 5. Experimental Setup

### 5.1 QuickTest Professional (QTP) 9.0

QTP is an automated functional Graphical User Interface (GUI) testing tool created by the HP subsidiary Mercury Interactive that allows the automation of user actions on a web or client based and desktop computer application. It is primarily used for functional regression test automation. QTP uses a scripting language built on top of VBScript to specify the test procedure, and to manipulate the objects and controls of the application under test. As part of a functional test suite, it works together with Mercury Interactive WinRunner and HP Quality Center and supports enterprise Quality Assurance.

QuickTest Professional 9.0 as an automated functional testing tool for different environments. You will use Quick Test Professional's graphical point and click interface to record and play back tests, add synchronization points and verification steps, as well as create multiple action tests. One can build upon fundamental topics by using debug tools to troubleshoot tests and use additional checkpoints and product options to broaden the scope of business processes that can be automated. Once tests are created, it can discover and correct common record and play back problems.

Automated testing with QuickTest addresses the problems by dramatically speeding up the testing process. One can create tests that check all aspects of your application or Web site, and then run these tests every time your site or application changes. As Quick Test runs tests, it simulates a human user by moving the cursor in a Web page or application window, clicking GUI (graphical user interface) objects, and entering keyboard input; however, Quick Test does this faster than any human user.

### 5.2 Winrunner 7.6

Mercury **WinRunner** offers an organization a powerful tool for enterprise-wide functional and regression testing. Mercury WinRunner captures, verifies, and replays user interactions automatically to identify defects and ensure that business processes work flawlessly upon deployment and remain reliable.

Mercury WinRunner's intuitive recording process allows us to produce robust functional tests. To create a test, WinRunner simply records a typical business process by emulating user actions, such as ordering an item or opening a vendor account. During recording, we can directly edit generated scripts to meet the most complex test requirements. Next, testers can add checkpoints, which compare expected and actual outcomes from the test run. Mercury WinRunner offers a variety of checkpoints, including test, GUI, bitmap, and web links. WinRunner can also verify database values to ensure transaction accuracy and database integrity, highlighting records that have been updated, modified, deleted, and inserted.

With a few mouse clicks, Mercury WinRunner's Data Driver Wizard lets us convert a recorded business process into a data-driven test that reflects the unique, real-life actions of multiple users. For further test enhancement, the Function Generator presents a quick and reliable way to program tests.

As Mercury WinRunner executes tests, it operates the application automatically, as though a real user were performing each step in the business process. If test execution takes place after hours or in the absence of a QA engineer, WinRunner's Recovery Manager and Exception Handling mechanism automatically troubleshoot unexpected events, errors, and application crashes to ensure smooth test completion. Once tests are run, Mercury WinRunner's interactive reporting tools help your team interpret results by providing

detailed, easy-to-read reports that list errors and their origination. WinRunner enables your organization to build reusable tests to use throughout an application's lifecycle. Thus, if developers modify an application over time, testers do not need to modify multiple tests. Instead, they can apply changes to the GUI Map, a central repository of test-related information, and WinRunner will automatically propagate changes to all relevant scripts.

## 6. Calculation For Metrics

The values of the metrics are been calculated for the comparison of the effectiveness of software testing tools. For the help in calculating the same some of the programs are selected.

### 6.1 Tool Management

| | QTP 9.0 | WIN RUNNER 7.6 |
|---|---|---|
| AL | 2 | 2 |
| ICM | 3 | 2 |
| TM | 5 | 4 |

### 6.2 Test Case Generation

| | QTP 9.0 | WIN RUNNER 7.6 |
|---|---|---|
| ATG | 8 | 6 |
| TRF | 10 | 8 |
| TCG | 18 | 14 |

### 6.3 Maturity and Customer Base

| | QTP 9.0 | WIN RUNNER 7.6 |
|---|---|---|
| M | 7 | 13 |
| CB | 9 | 5 |
| P | 17 | 8 |
| MCB | 33 | 26 |

### 6.4 Tool Support

| | QTP 9.0 | WIN RUNNER 7.6 |
|---|---|---|
| ART | 1.5 sec | 2.2 sec |
| ARTAH | 180 sec | 190 sec |
| ATSD | 1 sec | 2 sec |
| DI | 0 | 0 |
| TS | 182.5 | 194.2 |

### 6.5 Response Time

- The tool QTP 9.0 works well with regards to response time. It takes on an average 6 minutes to perform.
- The tool WIN RUNNER 7.6 works well with regards to response time. It takes on an average 9 minutes to perform.

### 6.6 Feature Support

The features supported is the count of following features:

1) Tool supports user written functions extending tool functionality

2) Stores information in a data base open to the user

3) Integrates itself to the software development tools.

**QTP 9.0** scores "3" by supporting the all three features. **WIN RUNNER 7.6** scores" 2.5" by supporting first two features fully and third feature partially.

## 7. Future Scope & Conclusion

The further extension of the research is possible by generating the metrics which can be applicable to all types of testing tools which can differentiate the automated testing tools and suggest the best for the right application. Another avenue of future research is to generate the metrics which can help to compare the testing tools working under various operating system configurations and tools setting.

The metrics proposed explains the characteristics of two automated testing tools by collecting the data. The different scores of data collected on the basis of sub fields of metrics directs towards the right choice by comparing the data scores .The result reflects that as per the metrics the QTP is far better than the Winrunner testing tool. The metrics are capable to distinguish the difference in between the Blackbox testing tools relative to the software system under test .This theory is supported by the industry as Winrunner is already outdated and suggested by the vender to get the tool replaced by the QTP.

## References

[1] Jaana Lindroos, Helsinki," Code and Design Metrics for OOS", Seminar on Quality Models for Software Engineering. 1st Dec 2004.
[2] Paakki Jukka, Verkamo Inkeri, Gustafsson Juha, Nenonen Lilli," Metrics for Analysis and Improvement of Software Architectures (MAISA)",Research and Development Project at the University of Helsinki in Department of Computer Science, 1999 – 2001.
[3] Dekkers C.,"Demystifying function points Let's understand some terminology" IT Metrics Strategies, Oct. 1998.
[4] Mercury: WinRunner product overview, ttp://www.mercury.com/de/products/ qualitycenter/functional-testing/winrunner/, (last visited: July 2005).
[5] McCabe, T. J.,"A complexity measure", IEEE Trans. Software Eng. SE-2, 4 (Dec.1976), 308-320.
[6] Chidamber, S. R. and Kemerer, R. F.,"A metrics suite for object-oriented design", IEEE Trans. Software Eng. 20, 6 (June 1994), 476-493.
[7] Daich, G T Price, G Ragland Dawood M Utah," Software Test Technologies", Aug 1994.
[8] Postan, R M and Sexton,"Evaluating and Selecting Testing Tools",IEEE Software May 1992, 33-42.
[9] Shuqin Li-Kpkko,"Code and Design Metrics for Object-Oriented Systems",Helsinki University of Technology, 2000.
[10] Beck Kent,"Extreme Programming Explained:Embrace change",Addison-Wesley, 2001.
[11] Lionel C.,Briand,Sandro Morasca,Victor R. Basili,"defining and Validation Measure for Object-Based High-Level Design", Fellow, IEEE,1999.
[12] Alkadi Ghassan,Carver Doris L.,"Application of Metrics to Object-Oriented Designs",Proceedings of IEEE Aerospace conference, 1998.
[13] Halstead, M. H., "Elements of Software Science", NewYork Elsevier Science, 1977.
[14] Neville I. Churcher,Martin J. Shepperd,"Comments on'A Metrics Suite For Object Oriented Design",IEEE Transaction on Software Engineering, 1995.
[15] Ping Yu,Tarja Systa,Hausi Miiller,"Predicting Fault-Proneness Using OO Metrics An Industrial Case study",Proceedings of the Sixth European Conference on Software Maintenance and Reengineering, IEEE, 2002.
[16] Lorenz Mark,Kidd Jeff,"Object-Oriented Software Metrics",A Practical Guide,1994.
[17] Archer Clark,Stinson Michael,"Object-Oriented Software Measures",Technical Report CMU/SEI-95-TR-002,Software Engineering Institute,Camegie Mellon University, 1995
[18] Gamma Erich,Helm Richard,Jahnson Ralph,Vlisside John,"Design Patterns CD:Elements of Reusable Object-Oriented Sofrware",Addison Wesley Longman, 1998.
[19] P.G. Frankl and E.J. Weyuker,"An Analytical Comparison of the Fault-detecting Ability of Data Flow Testing Techniques",Proceedings Fifteenth International Conference on Software Engineering,Baltimore, Md., May 1993.
[20] I. Pomeranz and S.M. Reddy,"Properties of Maximally Dominating Faults",IEEE Proc. On ATS'04, Dec. 2004, pp.
[21] Metrics for Quantification of the Software Testing Tools Effectiveness. American Journal of Software Engineering and Applications 2015; 4(1): 15-22 Published online April 14, 2015.