American Journal of Software Engineering and Applications
2015; 4(1): 15-22
Published online April 14, 2015 (http://www.sciencepublishinggroup.com/j/ajsea)
doi: 10.11648/j.ajsea.20150401.12
ISSN: 2327-2473 (Print); ISSN: 2327-249X (Online)

Metrics for Quantification of the Software Testing Tools Effectiveness

Pawan Singh*, Mulualem Wordofa Regassa

School of Informatics, IOT, Hawassa University, Awassa, Ethiopia

Email address:
pawansingh3@yahoo.com (P. Singh), dr_pawansingh@hu.edu.et (P. Singh), jimpowerdire@gmail.com (M. W. Regassa)

To cite this article:
Pawan Singh, Mulualem Wordofa Regassa. Metrics for Quantification of the Software Testing Tools Effectiveness. *American Journal of Software Engineering and Applications.* Vol. 4, No. 1, 2015, pp. 15-22. doi: 10.11648/j.ajsea.20150401.12

Abstract: An automated testing tool helps the testers to quantify the quality of software by testing the software automatically. To quantify the quality of software there is always a requirement of good testing tools, which satisfy the testing requirement of the project. Although there is a wide range of testing tools available in the market and they vary in approach, quality, usability and other characteristics. Selecting the appropriate testing tool for software there is a requirement of a methodology to prioritize them on the basis of some 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 tedious part of the software development process. There are lots of different automated software testing tools currently available in the market. Some of these tools are only able to perform specific kind of testing and other products support a wide range of applications and offer more features and functionality. Automated testing tools help the testers to quantify the quality of software by testing the software. To quantify the quality of software, there is always a requirement of appropriate testing tool, which satisfy the testing requirement of the project. 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. Through this paper we proposed a set of metrics for measuring the characteristics for examination and selection of automated testing tool. This set of metrics will help to quantify the quality of automated software tools. Using this set of metrics the evaluation of effectiveness of testing tools can be done. With the help of the metrics the comparison of characteristics of different available testing tools can be performed to select the best suited for the corresponding project. In the development process the evaluation of software testing tools effectiveness has become an important factor to be considered for software testing and assessment, especially for critical software. A new extended model which is proposed provides the metrics to calculate the effectiveness of functional testing tool on the basis of operability. Students are studying only the traditional metrics to evaluate the software quality but this set of metrics will help them to understand the fundamentals of selection of exact and suitable tool for software testing. The industry will also be benefited; they can use the metrics to evaluate tools and reduce the testing effort hence saving time and extracting maximum benefits.

2. Metrics

In software engineering, any sort of quality can be quantified in terms of metrics. Software metric a measurable property, is an indicator of one or more of the quality criteria that we are seeking to measure. 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. It was assumed that more line of codes implied more complex programs, which shows a possibility of having more

errors. However software metrics have evolved well beyond the simple measures introduced in the 1960s.

2.1. Traditional Metrics

The traditional metrics are those which is been taught to the students from a long period of time and it only quantifies the quality of the software.

2.1.1. Cyclomatic Complexity (CC)

It measures the amount of decision logics in a single software module. The use of CC [1] is in two related purposes in the structured testing methodology. First, it provides the number of optional tests for software. Second, the use of CC is during all phases of the software development lifecycle, starting with design, to maintain software reliable, manageable and testable. The structure of software's control flow graph is the basis of Cyclomatic complexity. The word "cyclomatic" derives from the number of fundamental or basic cycles in connected, undirected graphs. More essentially, CC also provides the number of independent paths through strongly connected directed graphs. In strongly connected graph each node can be reached from any other node by following directed edges in the graph. The cyclomatic number in graph theory is defined as:

$$CC = e - n + 2P \quad (1)$$

Program control flow graphs (CFG) are not strongly connected, but they become strongly connected by adding a virtual edge connecting the exit node to the entry node. The CC definition for program control flow graphs is resultant from the cyclomatic number formula by merely adding one to represent the contribution of the virtual edge. According to this definition the cyclomatic complexity equals the number of independent paths through the standard control flow graph model, and avoids explicit declaration of the virtual edge.

$$M = V(G) = e - n + 2P \quad (2)$$

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)

It is a metric that may be applied independent of a specific programming language, in fact, it can be determined in the design stage prior to the commencement of coding. To determine FP, an Unadjusted Function Point Count (UFP) is calculated [2]. UFP is found by counting the number of external inputs (user input), external outputs (program output), external inquiries (interactive inputs requiring a response), external files (inter-system interface), and internal files (system logical master files). Each member of the above five groups is analyzed as having either simple, average or high complexity, and a weight is associated with that member based upon a table of FP complexity weights. UFP is then calculated via:

$$UFP = \sum_{i=1}^{15}(\text{number of items of variety i}) * (\text{weight of i}) \quad (3)$$

Next, a Complexity Adjustment Factor (CAF) is determined by analyzing fourteen contributing factors. Each factor is assigned a score from zero to five based on its criticality to the system being built. The CAF is then found through the equation:

$$CAF = 0.65 + 0.01 \sum_{i=1}^{14} Fi \quad (4)$$

FP is the product of UFC and CAF. FP has been criticized due to its reliance upon subjective ratings and its foundation on early design characteristics that are likely to change as the development process progresses.

2.1.3. Halstead

Halstead [3] created a metric founded on the number of operators and operands in a program. His software-science metric (a.k.a. halted length) 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 (i.e., program complexity), program difficulty, potential minimum volume of an algorithm and other measurements.

2.2. Object Oriented Software Metrics

Object-oriented design and development has turn out to be popular in present software development environment. There exists wide recognition of benefits of object-oriented software development. Object-oriented development needs not only diverse approaches to design and implementation; it also requires different approaches to software metrics [4]. The metrics for object-oriented systems are different from structured system due to the difference in program paradigm and language itself [5]. An object-oriented program paradigm and structure are different from procedural languages; it uses localization, inheritance, information hiding, encapsulation, object abstraction and polymorphism. There are quite a few sets of proposed metrics for object-oriented software in the literature. The definition of six different metrics is specified in this text.

2.2.1.Weighted Methods per Class (WMC)

WMC is a sum of complexities of methods of a class. Consider a Class C1 with Methods $M_1 \ldots M_n$ that is defined in the class. Let $c_1 \ldots c_n$ be the complexity of the methods. Then:

$$WMC = \sum_{i=1}^{n} c_i \quad (5)$$

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 a class. The bigger the count of methods within a class larger the potential impact on inheriting classes. Consequently, more effort and time will be needed for maintenance and testing. Furthermore, classes with large number of complex methods are likely to be more application specific, limiting the possibility of reuse [4], [7],

[8], [9]. Estimation of usability and reusability of the class can be done using WMC. If all method complexities are considered to be unity, then WMC equals to Number of Methods (NMC) metric.

2.2.2. Depth of Inheritance Tree (DIT)

The depth of a class inside the inheritance hierarchy is the largest length from the class node to the root of the tree, calculated by the number of ancestor classes [4], [8] and [10]. The deeper a class is in the hierarchy, the larger the number of methods it is likely to inherit, making it more difficult to forecast its behavior. Deeper trees constitute higher design complexity, since more methods and classes are concerned. The deeper a particular class is in the hierarchy, the larger prospective reuse of inherited methods. The longest path is usually considered for languages that allow multiple inheritances. The large DIT is as well associated to understandability and testability. The complexity can be decreased using inheritance by reducing the count of operations and operators, but this abstraction of objects can create maintenance and design complicated.

2.2.3. 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 [4], [7], [8], [9]. If greater the number of children than more is likelihood of improper abstraction of the parent class. 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. In addition, a class with a large number of children must be flexible in order to provide services in a large number of contexts.

2.2.4. 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, i.e., methods of one uses methods or instance variables of another [4], [9], [10], [11]. 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. In order to improve modularity and promote encapsulation, inter-object class couples should be kept to a minimum. Direct access to foreign instance variable has generally been identified as the worst type of coupling. The larger the number of couples, the higher the sensitivity to changes in other parts of the design, and therefore maintenance is more difficult. A measure of coupling is useful to determine how complex the testing of various parts of a design is likely to be. The higher the inter-object class coupling, the more rigorous the testing needs to be.

2.2.5. Response for a Class (RFC)

The response set of a class is a set of procedures that can potentially be executed in response to a message established by and object of that class. RFC evaluate both external and internal communication, but particularly it comprises procedures called from outside the class, so it is also a gauge of the potential communication between the class and other classes [4], [8], [9]. RFC is more perceptive evaluation of coupling than CBO since it take into account procedure instead of classes. If a huge count of procedures can be invoked in account to a message, the testing and debugging of the class turn out to be more problematical since it requires a greater level of perceptiveness required on the part of the tester. The greater the count of routines that can be invoked from a class, the greater the complication in the class. A worst-case value for possible reaction will assist in suitable allocation of testing time.

2.2.6. Lack of Cohesion in Methods (LCOM)

The LCOM is a calculation of the number of procedure pairs whose resemblance is “0” minus the count of procedure pairs whose resemblance is non zero. The larger the count of number of alike procedures, the greater cohesive the class, which is consistent with conventional notions of cohesion that calculate the inter-relatedness between segments of a program [4], [7], [10]. If none of the procedure of a class reflects any instance performance, i.e., do not use any instance variables, they consists no resemblance and the LCOM value for the class will be zero. Cohesiveness of procedures inside a class is desirable, since it encourages encapsulation. Short of cohesion indicates classes should most likely be divided into two or more subclasses. Any calculation of disparateness of procedure helps recognize flaws in the design of classes. Low cohesion augments complication; thereby it increases the probability of errors at some stage in the development process.

3. Prior Work on Metrics

The Institute for Defense Analyses (IDA) made available two survey information on tools for testing software. Although the tool explanations provided in those reports are dated, the examination provide a historical frame of reference for the current progress in testing tools and recognize a great count of quantity that may be utilized in evaluating testing tools [12], [13]. For every tool, the report specifies diverse types of analysis carried out. Software Technology Support Center (STSC) works with Air Force software organizations to recognize assess and accept technologies to advance product quality, augment production effectiveness, and hone cost and schedule forecast capability [14]. Section four of the report tells about numerous issues that should be addressed when examining testing tools and offers a model tool-scoring matrix.

Brett Daniel [6] has provided a broad summary of the process he suggested to calculate effectiveness of an automatic test production tool. The method has two parts: instructing a decision tree by means of code with identified coverage characteristics and using the tree to forecast coverage on new code. It starts with a big amount of source code, which we say as the training code. It provides two data sets from the training

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 in Evaluation and repair tools [15]. His examination of six accessibility testing tools was focused at calculating the cost/benefit ratio and supporting probable customers to choose the most suitable tool. In submission to allowing for costs, availability, and accuracy, the examination scope was quality of use. The procedure he used is highly influenced by manual and methodical inspection of the results formed by the tools on chosen test pages, and is consequently less generally relevant than the procedure suggested in his paper, as it needs carefully generated test files and a lengthy and subjective analysis of the results.

In a current paper, Ivory and her colleagues focused at examining quality of use of testing tools [16]. They carry out an experiment where web designers were required to use testing tools and to amend web sites as a result to what tools suggested. Afterward in second experiment, the authors extract how effective such modifications were for disabled web site visitors.

The criteria proposed by Poston and Sexton [17] aimed on company precise criteria or on criteria demanding an intense effort to be evaluated e.g. test effort or test quality, these criteria do not narrate for a pre-selection of the test tools. In addition to it, criteria precise to test tools is stated without a validation for their derivation. The criteria stated by Poston and Sexton signify a subset of the criteria methodologically derived in our approach.

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

These metrics are used to quantify the effectiveness of testing tools on the basis of the capability of and ease to operate. The testing tools are desired to be easy to use in all terms and it must be popular if it is giving good results in terms of working. The vendor's responsibility includes making it simple and informative as well. On the basis of some of the operational properties there are few metrics provided by us which can help the tester to select the appropriate tool for his projects:

4.1.1. Toughness of Interface (ToI)

To start testing with automated software testing tool one need to configure it first. If any tool is designed in a proper manner than the human interface with the tool will lead to simple, efficient and correct setting of tool configuration but if the design is inadequate then the number of keyboard to mouse switches will be large, number of input fields provided will be large with long input strings and required output fields will also be large in number.

$$\text{ToI} = \frac{1}{\sum \text{F}}\left[\sum\left(\frac{\text{SKM}}{\text{t}}\right) + \sum\left((\text{IF} * \text{TLIF}) + \text{OF}\right)\right] - \text{BBF} \tag{6}$$

In (1) SKM / t is the number of switches from keyboard to mouse per unit time, IF is the average number of input fields, TLIF is the total string length of input fields, OF is the number of output field required, BBF is the number of buttons based functions. The values of SKM, IF, TLIF, OF is calculated per function. A large value of ToI indicates the toughness in learning the tool due to complicated interface. This can also lead to possibility of errors if we use the tool for a long time. A tool with lesser value of ToI must be selected.

4.1.2. Customers Affection and Tool Age (CATA)

There exist a number of designers of automated testing tools carrying different approaches and experiences. If the tool is properly designed and having good customer satisfaction of testing their software on the tool, then this satisfaction level can be used to motivate other users to use that testing tool. The maturity of a software counts to suggest the same tool as it shows customers having trust on that tool from a long period of time.

$$\text{CATA} = \text{CA} + \text{TA} \tag{7}$$

In (7) CA customer affection is the number customers of that tool and they are using tool from more than one year. TA tool age is the number of years the tool is in use including its previous versions. A tool with a larger value of CATA is selected for testing software.

4.1.3. Projects Handled (PH)

There In making the decision to select a testing tool for a software of known expected size the major factor will be the experience of tool in dealing with similar sort of project with same size and different size. The tool having good experience of dealing with similar size will be the preferred one but the tool having experience in dealing with big in size project will also be better than the tool dealing with small projects.

$$\text{PH} = \alpha * \text{STSS} + \beta * \text{STBiS} + \gamma * \text{STSiS} \tag{8}$$

In (8) $\alpha < \beta < \gamma$, STSS is the number of projects of same type and same size tested previously on that tool, STBiS is the count of projects with same type and big in size tested previously on that tool and STSiS is the count of projects with same type and small in size tested previously on that tool. A tool having a large value of PH is the suggested.

4.1.4. Inconvenience of Use (IU)

An automated testing tool is desired to be easy in terms of using it first time and in subsequent attempt. The tool which is better to learn in first will be given preference to the tool easy to use in subsequent attempts. The desired property of a testing tool is its retainability of use means the procedure of using can be easily remembered by either a casual user or by the frequent user. If the time required using a tool for testing by either a casual user or frequent user is less it will be the preferred one.

$$IU = (0.75 * LTFTU + 0.25 * LTExUPV) – (RoPc + RoPf) + (AOTc + AOTf) \quad (9)$$

In (9) LTFTU is the learning time for first users in days, LTExUPV is the learning time for the experienced user of previous version in days, RoPc is the retainability of procedure knowledge for casual users, RoPf is the retainability of procedure knowledge for frequent users, AOTc is the average operational time for casual users in hours and AOTf is the average operational time for frequent users in hours. The tool having a less value of IU is better.

4.1.5. Documentation Support (DS)

To have the efficient implementation and use of tool a proper documentation must be provided. A user of a tool may need certain information at different time for reference this information is usually provided by the vendor in the form of documentation. The tools effectiveness is influenced by the time required to search some information and by the number of ways the information can be accessed. It is negatively supported if the documentation is inadequate.

$$DS = WIA – (ID + ASTD) \quad (10)$$

In (10) ASTD is the average search time of documentation, WIA is the number of ways the information can be accessed and ID stands for the inadequacy in documentation which is measured as the number of unsuccessful searches of documentation. The higher value of DS is expected for a good tool.

4.1.6. Tools Reliability (TR)

Tool reliability of any system is defined as inverse of number of failure per unit time. Lesser the value of number of failure better the tool is as reliable tool.

$$TR = \frac{1}{N/T} \quad (11)$$

In (11) TR is tool reliability, N is the number of failure of testing software and T is the total time. It is expected that the tool must be with high value of TR.

4.2. Functional Metrics

These metrics are used to quantify the effectiveness of testing tools on the basis of the capability of testing the software programs. The operational simplicity and informativeness are not the only factors which must be included in the decision to select the testing tool but there are some more functional properties required to be included. Functional metrics incur some cost in terms of some sort of pre testing required to analyze the tools. The functional testing may be better utilized to prepare the data regarding the tools for the further projects to be evaluated. On the basis of functional properties there are some metrics provided by us which can help the tester to select the appropriate tool for his projects:

4.2.1. Tool's Completeness (TC_m)

Tool's Completeness is a computation of how many accessibility defects present in the software are noticed and revealed to the user. Completeness is associated to how well the tool is capable to seize defects. Completeness is a complex property to distinguish operationally. In fact it requires knowing the true problems in advance. Therefore determining the true problems means accurate usability investigations through some means of testing.

$$TC_m = DCS/TDP \quad (12)$$

In (12) DCS is the number of defects of software caught and shown by the tool to the user. TDP is the total number of defects actually present in the software. The value of TC_m is required to be high for a better selection.

4.2.2. Tool's Correctness (TC_r)

Tool's Correctness is the fraction of problems reported by the tool that are certainly true problems and the actual total number of defects actually present in the software. Correctness is associated to how well a tool reduces actual defects. It requires the potential and knowledge of true and false defects shown by the tool otherwise it may lead to wrong decision.

$$TC_r = (DCS - FDCS)/TDP \quad (13)$$

In (13) DCS is the number of defects of software caught and shown by the tool to the user. FDCS stands for false defects of software caught and shown by the tool to the user. TDP is the total number of defects actually present in the software. TCr value must be bigger for a tool to be selected.

4.2.3. Tool's Coverage (TC_v)

Tool's Coverage is defined as the number of different types of defects possible to detect and described by a tool. The larger this set the more capable is the tool of providing specific warnings and suggestions, and therefore the more useful it is for the developer. It is not an easy property to be determined and not necessarily related to the tool effectiveness. It is good to have high value of TC_v.

5. Experimental Setup

To validate our proposed set of metrics for examination and selection of software testing tools, we have selected two automated software testing tools QTP 9.0 and WinRunner 7.6 to apply our metrics. In the following section we have described the working of these tools.

5.1. Quick Test Professional 9.0 (QTP 9.0)

It is an automated functional Graphical User Interface based testing tool generated by HP subsidiary Mercury Interactive that permit the automation of user actions on the web or client based and desktop computer application. It is principally used for functional regression test automation QTP requires a scripting language built on top of VBScript to indicate the test method and to update the object and control of application under test.

As a part of functional test suite, it performs together with Mercury Interactive Winrunner and HP Quality Centre and support project Quality Assurance.

Quick Test Professional 9.0 is an automated functional testing tool for diverse environments. It is having graphical point and click interface to record and play tests, add synchronization points and verification steps as well as create multiple action tests. As Quick Test runs test it simulates a human user by moving the cursor in a webpage or application window, clicking GUI objects and entering keyboard inputs; however Quick Test does this faster than any human.

5.2. WinRunner 7.6 (WR 7.6)

It offers an organization a power full tool for enterprise-wide functional and regression testing. Mercury WinRunner captures, verifies and replay user interactions automatically to identify defects and ensure that business processes work flawlessly upon deployment and remains reliable. Its intuitive recording process allows us to produce robust functional tests.

To create a test it simply records a typical business process by emulating user actions, such as ordering an item or opening a vender account. It executes tests and operates the application automatically, as though a real user is performing each step in business process. Its interactive reporting tool helps us interpret results by providing detailed, easy to read report that lists errors and their origination. It enables to build reusable tests to use throughout an application lifecycle.

For validation of our metrics we have selected 1047 small projects (codes) for testing using the two tools. For every code we have generated 20 to 73 test cases depending on the size of code and its complexity. The sizes of codes are ranging from 37 to 109 lines of codes.

In calculating the metrics the average values of the factors involved in the metrics are considered. The values calculated for different factors and finally the metrics are shown in the tables 1 to 7.

Table 1. *Calculation for ToI.*

Factors	QTP 9.0	WR 7.6
$^{1}/_{\sum F}\sum(SKM/t)$	4.3	6.4
$^{1}/_{\sum F}\sum(IF * TLIF)$	32	47
$\sum OF$	2	2
BBF	6	5
ToI	32.3	50.4

In the tables from 1 to 7 different factors are calculated and the metrics for both the tools QTP 9.0 and WR 7.6. In table 1 ToI metric value for QTP is less means it is less tough in using its interface by all types of users.

Table 2. *Calculation for CATA.*

Factors	QTP 9.0	WR 7.6
CA	25	15
TA	14	19
CATA	39	34

In table 2 the tool age of WR is shown more even then the customer of QTP is more than the WR so the overall value of CATA is greater and reflects preference of QTP.

Table 3. *Calculation for PH.*

Factors	QTP 9.0	WR 7.6
α * STSS	7.5	3
β * STBiS	2	1.25
γ * STSiS	.5	1
PH	10	5.25

As it is suggested that the project experience of same type and same size is better than the project of same type and big in size as well as project of same type and small in size so the values considered of α= .5, β= .25 and γ= .25. This gives the PH high value (as shown in table 3) to QTP hence upper hand with respect to WR.

Table 4. *Calculation for IU.*

Factors	QTP 9.0	WR 7.6
0.75 * LTFTU	5.25	7.5
0.25 * LTExUPV	.75	.75
RoP_c	.25	.20
RoP_f	.78	.66
AOT_c	.75	.75
AOT_f	.33	.416
IU	6.05	8.556

The high value of IU indicated in table 4 for WR shows that it not easy to learn, operate and remember the working of WR than QTP.

Table 5. Calculation for DS.

Factors	QTP 9.0	WR 7.6
WIA	4	4
ID	0	0
ASTD	1	2
DS	3	2

As per the value of DS computed in the table 5 it is clear that the information provided and ease in access of information is better supported in the QTP than WR so any one will wish to have QTP for his project.

Table 6. *Calculation for TR.*

Factors	QTP 9.0	WR 7.6
N	3	3
T	2	2
TR	.66	.66

The statistics of table 6 indicates that both tools are similarly good in terms of reliability.

Table 7. *Calculation forTC_m AND TC_r.*

Factors	QTP 9.0	WR 7.6
TDP	76	76
DCS	61	56
FDCS	3	4
TC_m	.8026	.7368
TC_r	.7631	.6842

The high values computed for QTP in table 7 for the metrics tools completeness and tools correctness gives an idea of better functionality of QTP and it suggest to use QTPunless until not specified or mandatory for tester to use WR.

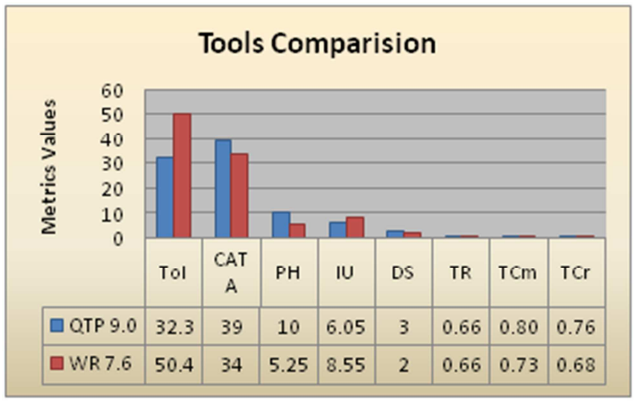

Figure 1.*Comparison of metrics for tools QTP and WR.*

From Fig. 1 it is shown that the metrics computed for both the tools indicates the high hand of QTP over WR.

6. Summary and Future Scope

In the current trend most of the software is required to be evaluated for two reasons. One the customer wants to get satisfied from the quality of the product which he will be going to use because he has invested a lot of amount to get the product. Two the developers want to quantify the quality so that his effort in further stages and in maintenance must not increased due to low quality. The testing is required to be fast and with less effort that is through the automated testing tools. The setback is confusion of way of selection of tool. The positive side of the paper is that it helps the developer in deciding the best testing tool as per his project by calculating the metrics value for the available tools in hand. The weakness of the proposed work is that it required a bit of time and effort in doing calculations for evaluating the tools and some data may also be needed in calculation of metrics.

As per the experience we have during the work, we would like to mention our view for the further extension of this work. The upper and lower bound of the tools metrics must be investigated. The more number of tools must be exposed to metrics to have the further empirical analysis. The categorization of the tools on the basis of functionality is required and the specification of particular metric for that type may be suggested. An algorithmic approach may be generated to have the fast automated evaluation of metrics which may reduce the effort to calculate the metrics and automated suggestion for the better tools may also be a work piece.

7. Conclusion

To evaluate a tool it is highly required to check its functionality and operational potential. In our presented work the metrics are working on the operational and functional factors. The operational factors indicate the capability and easiness of handling the tool. The functional metrics are the reflection of ability of tool in tackling the software testing, its function. The operational metrics are almost static in nature as they can be calculated without performing any testing on tools to be selected, whereas the functional metrics needed some basic efforts to evaluate the tool's effectiveness.

In our work, the metrics proposed are applied on two tools QTP and WinRunner, here by comparing the calculated values of the metrics for the tools we came to have two conclusion. As per the first conclusion QTP is better tool with respect to WinRunner. The second conclusion is about our metrics which clearly discriminate the tools on different basis and these metrics are useful for developers as well as researchers to quantify the effectiveness of tools to get the help in decision of tool's selection. Using these metrics will help the tester to select the appropriate tool for his project it will save his time and removes his confusion.

References

[1] J. T. McCabe, "A complexity measure," IEEE Trans. Software Eng. SE-2, 4, pp. 308-320, Dec 1976.

[2] C. Dekkers, "Demystifying Function point: Lets understandsome terminology," IT metrics strategies, Oct 1998.

[3] M. H. Halested, "Elements of software science, " New York: Elsevier Science, 1977.

[4] S. R. Chidamber and R. F. Kemerer, "Ametrics suite for object-oriented design," IEEE Trans. Software Eng.vol. 20, 6, pp. 476-493,June1994.

[5] W. Li and S. Henry, "Object oriented metrics that predicts maintainability," Journal of System and Software, vol. 23, 2, pp. 111- 122, Nov 1993.

[6] B. Daniel and M. Boshernitsan, "Predicting and explaining automated testing tool effectiveness," University of Illiois at Urban- Campaign, Tech. Rep. UIUCDCS-R-2008-2956, April 2008.

[7] M. Lorenz and J. Kidd, "Object Oriented Software Metrics," Printice Hall Publishing, 1994.

[8] McCabe & Associates, McCabe Object Oriented Tool Usre's Instruction, 1994.

[9] Linda H. Rosenberg, "Metrics for Object Oriented Environment," EFAITP/AIE Third Annual Software Metrics Conference, December 97.

[10] R. Hudli, C. Hoskins and A. Hudli, “Software Metrics for Object Oriented Design,” IEEE, 1994.

[11] Y. Lee, B. Liang and F. Wang, “Some Complexity Metrics for Object Oriented Program Based on Information Flow,” Proceedings: CompEuro, pp. 302-310, March 1993.

[12] C. Youngblut and B. Brykczynski, “An examination of selected software testing tools: 1992,” IDA Paper, Inst. For Defense Analyses, Alexandria, Va., pp -2925, Oct. 1993.

[13] C. Youngblut and B. Brykczynski, “An examination of selected software testing tools: 1993,” Supp. IDA Paper, Inst. For Defense Analyses, Alexandria, Va., pp -2769, Dec. 1992.

[14] G. T. Daich, G. Price, B. Ragland, and M. Dawood, “Software test technologies report,” Software Technology Support Center, Hill AFB, Utah, Aug. 1994.

[15] J. Thatcher, “Evaluation and Repair Tools,” posted on http://www.jimthatcher.com, June 2002.

[16] M. Y. Ivory, R. R. Sinha and H. A. Hearst, “Empirically validated web page design metrics,” In Proceedings of the Conference on Human Factors in Computing Systems, pp. 53-60, New York, NY, ACM press, 2001.

[17] R. M. Poston and M. P. Sexton, “Evaluating and selecting testing tools,” IEEE Software, vol. 9, 3, pp. 33-42, May 1992.