International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Impact Factor (2012): 3.358

A Novel Approach to Metric Assessment, Productivity

Kishore K[1], Naresh E[2], Vijaya Kumar B P[3]

[1]M.Tech in Software Engineering, Department of Information Science and Engineering
M. S. Ramaiah Institute of Technology, Bangalore

[2]Research Scholar, Jain University, Bangalore and Assistant Professor, Department of Information Science and Engineering
M. S. Ramaiah Institute of Technology, Bangalore

[3]Professor and Head of Department, Department of Information Science and Engineering
M. S. Ramaiah Institute of Technology, Bangalore

Abstract: ***Metrics are significant indicator of the effectiveness of a software testing process. The key to effective measurement lies in the capability to evidently make out the goals to be accomplished and the issues to be tackled. The foremost step to ascertain test metrics is to identify the key software testing processes that can be objectively measured. This information can be used as the baseline to define the metric and to decide what information will be tracked, who will track the information and at which frequency. Then the processes needed to effectively track, calculate, manage and interpret the defined metrics must be implemented. In this paper we discuss software measurement and metrics and their fundamental role in software development life cycle. This paper focusing on software test metrics discusses their key role in software testing process and also classifies and systematically analyzes the various test metrics.***

Keywords: Software Measurement, Software Metrics, Metrics Classification, Software Test Metrics.

1. Introduction

Software measurement has become a key facet of good software engineering practice. All phases of development process are well know using measurement activities and it adds value and also keep us actively involved in development process. Measurement has at all times been primary to the progress to any engineering discipline. Software metrics is been used in making value based decisions as well as in risk evaluation.

Software testing is an essential element in the Software Development Life Cycle and can deliver outstanding results if done properly and effectively [3] and software measurement can play a key role in increasing the effectiveness of testing process. Evaluating the test activities will give clear insight into the sufficiency of the test process and the anticipated time to produce a software product that can meet excellence standards. Software metrics are used for assessment of the software development process and the quality of the resultant product [5]. Software metrics aid measurements for management decision making. Metric assessment with early detections and correction of problems make them important in software.

2. Software Metrics

Measurement covers quantitative assessments that usually use metrics and measures which can be used to directly establish attainment of numerical quality goals. Test metrics are a very prevailing risk management tool. Metric assessment helps us to measure the recent performance. Test metrics acts as an aiding agent that managers can use to recognize their current state and to prioritize their activities to decrease the risk of schedule over-runs on software releases.

Metrics are figured out from measures. Metrics make easy the feature of some particular characteristic. Metrics are measurements of diverse aspects of an attempt that help us determine whether we are moving ahead towards the goal of that endeavor. Metrics are typically dedicated by the subject area, in which case they are valid only within a certain domain and cannot be straightforwardly benchmarked or take to mean outside it. Basically, software metrics deals with the measurement of the software product and the process by which it is developed. They are quantifiable indices used to measure up software products, processes, or projects or to predict their outcomes. It is necessary that software Metrics is well defined before they are utilized;

With Software metrics, we can:

- Identifies risk areas
- Identifies areas of improvement and track development
- progress
- Identify with maintenance costs.
- Envisage development estimation and facilitates performance gap

3. Categories of Metrics.

Software Metrics can be classified under different categories From Commercial Perspective, Metrics can be classified into five modules to measure the quantity and quality of software.

- Technical Metrics are used to determine whether the code is well-structured, that manuals for hardware and software use are adequate, that documentation is complete, correct, and up to date. Technical metrics also describe the external characteristics of the system's implementation.

- Defect Metrics are used to determine that the system does not erroneously process data, does not abnormally terminate, and does not do the many other things associated with the failure of a software-intensive system.
- End-User Satisfaction Metrics are used to describe the value received from using the system.
- Assurance Metrics reflect specific revenues and expenditures associated with correcting software defects on a case-by-case basis. These metrics are influenced by the level of defects, willingness of users to come forth with complaints, and the willingness and ability of the software developer to accommodate the user.
- Status Metrics are used to assess perceived user satisfaction with the software and may generate the most value, since it can strongly influence what software is acquired. Reputation may differ significantly from actual satisfaction:

From significance perspective, Metrics can be grouped into two classes:

- Core Metric is a required metric that is essential to support solution delivery test management on systems development projects. Example: Percentage of requirements met [14].
- Non-Core Metric is an optional metric that can help to create a more balanced picture of the quality and effectiveness of test efforts. Example: Total number of defects by test phase [14].

From observation perspective, Metrics can also be categorized as:

- Primitive metrics are those that can be directly observed, such as the program size, number of defects observed in unit testing, or total development time for the project [15].
- Computed metrics are those that cannot be directly observed but are computed in some manner from other metrics. Examples of computed metrics are those commonly used for productivity, such as Lines Of Code(LOC) produced per person-month, or for product quality, such as the number of defects per thousand lines of code (defects/KLOC). Computed metrics are combinations of other metric values and thus are often more valuable in understanding or evaluating the software process than are simple metrics [15].

From measurement perspective, Metrics can be classified as:

- Direct measurement of an attribute of an entity involves no other attribute or entity. Direct measurement is assessment of something existing [13]. E.g. number of lines of code.
- Indirect/ derived measurement means calculation involving other attributes or entities by using some mathematical model (always contains a calculation of at least two metrics) [13].

E.g. defect density = no. of defects in a software product / total size of product.

More often software metrics is categorized in a much broader sense as:

- Process metrics are measures of the software development process, such as overall development time, type of methodology used, or the average level of experience of the programming staff. They can be classified as empirical, statistical, theory base and composite models.
- Product metrics are measures of the software product at any stage of its development, from requirements to installed system. Product metrics may measure the complexity of the software design, the size of the final program (either source or object code), or the number of pages of documentation produced. They are often classified according with the size, complexity, and quality and data dependency.

Software metrics can also be classified as:

- Objective metrics should always result in identical values for a given metric, as measured by two or more qualified observers.
- Subjective metrics may measure different values for a given metric, since their subjective judgment is involved in arriving at the measured value. For product metrics, the size of the product measured in lines of code (LOC) is an objective measure, for which any informed observer, working from the same definition of LOC, same measured value for a given program. An example of a subjective product metric is the classification of the software as “organic,” “semi-detached,” or “embedded,” as required in the cost estimation model [9].

Although software metrics can be neatly categorized as primitive objective product metrics, primitive subjective product metrics, etc., this module does not strictly follow that organization.

3.1 Characteristics of Usable Metrics.

The effective metrics can play a radical role in the change of performance of business. They are helpful in decision-making on a particular project or within a given organization and are helpful in resolving potential problems and also discover areas of improvement [13]. A functional metric is precisely defined (i.e., measurable), It also helps indicate whether an organization is achieving software goals [16]. There are several fundamental characteristics associated with useful software metrics.

- Simple and effortless to understand.
- assessable
- Cost-effective.
- Metrics must be well-timed.
- Consistent and applicable.
- Reliable and used over time.
- Precise.

In the following we can use the concrete criterion, which helps for checking if the previously stated research goals have been met. Well-definiteness, it is elementary to evaluate values obtained in different software systems over time to be definite about what they represent. it is crucial to define how we measure the values in the software systems.

Standard-based, this goal is achieve if the quality representation used is supported on an existing standard summarizing the state of the art/best performance, thereby gaining a high trust level. The metrics are selected from well known and discussed metrics.

Repeatability, this is to allow repetition of each single part of the testing and the experiment itself, in a well-organized way. This is essential since we cannot meet the expense of time consuming preparation. In general, this goal is accomplished if the complete data extraction, metrics estimation, and measurement procedure can be repeated in an efficient way, leading to the same results for first and third parties.

The criteria for this goal can be split into three sub decisive factor: Automation, Reusability and Maintainability.

Automation, It is feasible to extract data and compute metrics in an automated way. Thus, once defined series of data extractions and metrics computations can be functional in random order and occurrence on the same or different software systems, leading to the predictable results. The time used up on the initial definition is saved in successive uses, and user interface is abridged to a minimum or not needed at all. It is possible to incorporate the measurement process as a fully automated part into the software development process.

Reusability Metrics can be defined and execute in a generic way, supported by a meta-model. Briefly, a meta-model defines the rules and elements according to which models can be constructed.[31] This makes the metrics reusable to different software systems written in different programming languages. This means that, e.g., one and the same Number of Methods metric can be applied regardless if it is the number of methods in a class provided in Java Source Code or a class in a UML model. [31] The metrics are defined once only, and can be useful to software systems represented in any existing or potential software systems and languages, under the circumstance that the metric makes sense in that perspective at all.

Maintainability Mechanisms are provided, which allow metrics to be defined and implemented once in a generic way, so that they keep up with changes and extensions in the meta-model without becoming invalidated. E.g., a metric is defined upon a meta-model containing classes, it is then extended to distinguish classes and interfaces. The metric just expecting classes must not break, if it finds an interface. [31]

Identify the right metrics assists in ensuring that all the different class of metrics are considered based on the project requirements, enhancement of the goals or problem areas where upgrading is required.

4. Software Testing Metrics

The term used to describe a measurement of a particular attribute of a software project is a Software Metric. The Software Metrics are concerned with the test activities that are part of the Test Phase and so are formally known as Software Testing Metrics [17]. There are two broad categories of software metrics, namely product metrics and process metrics.

"Test process" metrics give information about preparation for testing, test execution and test growth. They are used to check the progress of testing, status of design and development of test cases and outcome of test cases after execution. Process metrics describe the effectiveness and quality of the processes that produce the software product. Examples are effort required in the process, time to produce the product, effectiveness of defect removal during development, number of defects found during testing, maturity of the process [18]. Some Test process metrics are:

1) Number of test cases designed.
2) Number of test cases executed.
3) Percentage of test cases passed/failed.
4) Total actual execution time / total anticipated execution time
5) Average execution time of a test case.

"Test product" metrics provide information about the test state and testing status of a software product and are generated by test execution and code fixes. Using these metrics we can measure the products test state and indicative level of quality, useful for product release decisions. Product metrics describe the characteristics of the product such as size, complexity, design features, performance, efficiency, reliability, portability, etc [11]. Some Test product metrics are:

1) Estimated time for testing.
2) Actual testing time.
3) Average time interval between failures.
4) Maximum and minimum failures experienced in any time interval.
5) Average number of failures experienced in time intervals.

Table 4 shows various test measures and resulting test metrics.

Testing Measures and Metrics	
Measure	*Metrics*
Test process efficiency	Resource allocation and utilization
Test productivity	Effort Variance, Time variance
Cost of Testing	Direct cost, Indirect Cost
Test Effectiveness	Defect Rejection, Defect Distribution

Table 4: Test Measures and Metrics

4.1. Importance of Software Testing Metrics

Software metrics are relevant to the whole development life cycle from commencement, when cost must be predictable to monitoring the reliability of the end product in the field, and the way that product revolutionize over time with improvement.[16] Software metrics cling to importance in testing phase, as software testing metrics acts as indicators of software quality and fault proneness. Testing metrics demonstrate trends and characteristics over time that would

be indicative of the stability of the process [18]. The essential step is establishing test metrics is to identify the key software testing processes that can be objectively measured. Measuring software development and testing projects is a challenging, but essential component of a professional organization. Software project may be running over time and over budget and still have a high number of defects. Or, it may be on time and on budget and have an even higher number of defects. Measuring allows you to quantify your schedule, development, and testing efforts. When you measure your current project performance, you become better equipped to schedule and budget for future projects. A major percentage of software projects suffer from quality problems, which in turn requires new testing metrics to measure test processes effectively. Test metrics are powerful risk management tool, help us to measure current performance.
Test metrics are key "facts" and serve following purposes [19]:

- Assists to understand the current position of the project.
- Provides a means for control/status reporting.
- Identify risk areas that require more testing.
- Test metrics provide an objective measure of the effectiveness and efficiency of testing.
- Setting quality benchmarks for several tasks and processes involved in development.

4.2. Notable Software Testing Metrics

Testing metrics may help us to measure the current performance of any project. The collected data may become historical data for future projects. This data is very important because in the absence of historical data, all estimates are just the guesses. Hence, it is essential to record the key information about the current projects. Test metrics may become an important indicator of the effectiveness and efficiency of a software testing process and may also identify risky areas that may need more testing.[15] Metrics that are associated with software testing are broken into three categories:

- Coverage: Meaningful parameters for measuring test scope and success.
- Progress: Parameters that help identify test progress to be matched against success criteria. Progress metrics are collected iteratively over time. They can be used to graph the process itself (e.g. time to fix defects, time to test, etc).
- Quality: Meaningful measures of excellence, worth, value, etc. of the testing product. It is difficult to measure quality directly; however, measuring the effects of quality is easier and possible.

5. General Uses of Metrics

- Software metrics are used to obtain objective reproducible measurements that can be useful for quality assurance, performance, debugging, management, and estimating costs. [17]
- Finding defects in code (post release and prior to release), predicting defective code, predicting project success, and predicting project risk [16]
- There is still some debate around which metrics matter and what they mean, the utility of metrics is limited to quantifying one of the following goals: Schedule of a software project, Size/complexity of development involved, cost of project, quality of software[16]

6. The Evaluation Framework

To evaluate a proposed metric, including one that we propose, we find it useful to ask the following useful questions:

1)What is the purpose of this measure? Examples of purposes include:
- Facilitating private self-assessment and improvement.
- Evaluating project status (to facilitate management of the project or related projects)
- Evaluating staff performance
- Informing others (e.g. potential customers) about the characteristics (such as development status or behavior) of the product
- Informing external authorities (e.g. regulators or litigators) about the characteristics of the product[19]

The higher the stakes associated with a measurement, the more important the validation.

2)What is the scope of this measure? A few examples of scope:
- A single method from one person
- One project done by one workgroup
- A year's work from that workgroup
- The entire company's output (including remote locations) for the last decade

As the scope broadens, more confounding variables can come into play, potentially impacting or invalidating the metric. A metric that works well locally might fail globally. [23]

3)What measuring instrument do we use to perform the measurement? Here are a few examples:
- Counting (by a human or by a machine). For example, count bugs, reported hours, branches, and lines of code.
- Matching (by a human, an algorithm or some other device). For example, a person might estimate the difficulty or complexity of a product by matching it to one of several products already completed. ("In my judgment, this one is just like that one.")
- Comparing (by a human, an algorithm or some other device). For example, a person might say that one specification item is more clearly written than another.
- Timing (by computer, by stopwatch, or by some external automated device, or by calculating a difference between two timestamps). For example, measure the time until a specified event (time to first failure), time between events, or time required to complete a task.[23]

A metric might be expressed as a formula involving more than one variable, such as Defect Removal Efficiency, (DRE) which is often computed as the ratio of defects found

during development to total defects (including ones found in the field).

4) What is the purpose of this measure? Bug counts have been used for a variety of purposes, including:
- Private, personal discovery by programmers of patterns in the mistakes they make. [22]
- Evaluation (by managers) of the work of testers (better testers allegedly find more bugs) and programmers (better programmers allegedly make fewer bugs).[23]
- Evaluation of product status and prediction of release date. [24]
- Estimation of reliability of the product. [26]

5) What attribute are we trying to measure? In software field, we've seen bug counts used as surrogates for quality of the product, effectiveness of testing, thoroughness of testing, effectiveness of the tester, skill or diligence of the programmer, reliability of the product, status of the project, readiness for release, effectiveness of a given test technique, customer satisfaction, even (in litigation) the negligence or lack of integrity of the development company. Some of the aspects of "goodness" of a tester employee are
- Skill—how well she does the tasks that she does. If we think of bug-hunting skill, we might consider whether the bugs found required particularly creative or technically challenging efforts),
- Effectiveness—the extent to which the tester achieves the objective of the work. For example, "The best tester isn't the one who finds the most bugs or who embarrasses the most programmers. The best tester is the one who gets the most bugs fixed." [27]
- Efficiency—how well the tester uses time. Achievement of results with a minimum waste of time and effort.
- Productivity - how much the tester delivers per unit time? The distinction that one can draw between efficiency and productivity is that efficiency refers to the way the person does the job whereas productivity refers to what she gets done. For example, a tester who works on a portion of the code that contains no defects can work through the tests efficiently but produce no bug reports.
- Diligence—how carefully and how hard the tester does her work.
- Courage—willing to attempt difficult and risky tasks; willing to honestly report findings that key stakeholders would prefer to see suppressed.
- Credibility—the extent to which others trusts the reports and commitments of this tester.

7. Conclusion

Software measurement and metrics assist us a lot in assessment of software process as well as the software product. The idea for the validation is to use static and dynamic metric as applied on the particular software process, and additional information sources like bug databases, and even human insights. They assist us in provide high quality and reliable software products. Software testing metrics are used to measure specific attributes of software product or processes during testing. Metrics alone will not progress testing, but they endow with information that will facilitate to focus and assess our product and process enhancement, but it is very important to know which metrics to use and which not to. In this paper we tried to cover software metrics which can improve testing significantly especially in terms of revelation, time and quality.

References

[1] Hofer, A. and Tichy, W. F. (2007). ,"Status of empirical research in software engineering.", In Empirical Software Engineering Issues, volume 4336/2007, pages 10–19. Springer.
[2] Aggarwal, K.K and Singh, Yogesh "Software Engineering Programs Documentation Operating Procedures (Second Edition)" New Age International Publishers, 2005.
[3] Quadri, S.M.K and Farooq, SU, "Software Testing Goals, Principles, and Limitations", International Journal of Computer Applications (0975 – 8887) Volume 6–No.9, September 2010.
[4] Munson, J. C. (2003). "Software Engineering Measurement". CRC Press, Inc., Boca Raton, FL, USA.
[5] Stark, George E; Durst, Robert C; and Pelnik, Tammy M. "An Evaluation of Software Testing metrics for NASA's Mission Control Center" 1992.
[6] Fenton, N. E. and Pfleeger, S. L., "Software Metrics: A Rigorous and Practical Approach", 2nd Edition Revised Boston: PWS Publishing, 1997.
[7] Stevens, S. S, "On the Theory of Scales of Measurement", Science, vol. 103, pp. 677-680, 1946.
[8] Stevens, S.S, "Psychophysics: Introduction to itsPerceptual, Neural, and Social Prospects", New York: John Wiley & Sons, 1975.
[9] Boehm, Barry w. (1981), "Software Engineering Economics", Englewood Cliffs, NJ. Prentice Hall.
[10] Farooq, SU and Quadri, S.M.K.,"Effectiveness of Software Testing Techniques on a Measurement Scale", Oriental Journal of Computer Science & Technology, Vol. 3(1), 109-113 (2010).
[11] Kan, Stephen H, "Metrics and Models In Software Quality Engineering", PEARSON, 2003
[12] Khelifi, Adel and Abran, Alain, "Software Measurement Standard Etalons: A Design Process", INTERNATIONAL JOURNAL OF COMPUTERS, Issue 3, Volume 1, 2007
[13] Futrell, Robert T.: Futrell ,Donald F. And Shafer, Linda I., "Quality Software Project Management", PEARSON
[14] Test Metrics, www.chakkilaminc.com
[15] Mills, Everald E, "Software Metrics SEI Curriculum module SEI – CM – 12 – 1.1¬, Carnegie Mellon University", Software engineering Institute, December, 1988.
[16] Torn, Aimo: Professor, Department of Computer Science Abo, Akademi University; Faculty member Turku Centre for Computer Science (TUCS) Turku, Finland.
[17] Kaur, Arvinder; Suri, Bharti and Sharma, Abhilasha,"Software Testing Product Metrics - A

Survey", Proceedings of National Conference on Challenges & Opportunities in Information Technology (COIT-2007) RIMT-IET, Mandi Gobindgarh. March 23, 2007.

[18] Ogasawara, Hideto, Yamada, Atsushi and Kojo, Michiko, "Experiences of software Quality Management Using Metrics through Life cycle", Proceedings of ICSE-18, 1996.

[19] Pusala, Ramesh "Operational Excellence through efficient Software Testing Metrics" Infosys, 2006.

[20] C. G. Hempel, Fundamentals of Concept Formation in Empirical Science: International Encyclopedia of Unified Science, vol. 2. Chicago: University of Chicago Press, 1952.

[21] D. Hoffman, "The Darker Side of Metrics," presented at Pacific Northwest Software Quality Conference, Portland, Oregon, 2000.

[22] W. Humphrey, "Introduction to the Personal Software Process." Boston:Addison-Wesley, 1996.

[23] C. Kaner, "Don't Use Bug Counts to Measure Testers," in Software Testing & Quality Engineering, 1999,

[24] E. Simmons, "When Will We be Done Testing? Software Defect Arrival Modeling Using the Weibull Distribution," presented at Pacific Northwest Software Quality Conference, 2000.

[25] S. H. Kan, J. Parrish, and D. Manlove, "In-Process Metrics for Software Testing," IBM Systems Journal, vol. 40, 2001.

[26] S. Brocklehurst and B. Littlewood, "New Ways to Get Accurate Reliability Measures," IEEE Software, vol. 9, 1992.

[27] C. Kaner, J. Falk, and H. Q. Nguyen, Testing Computer Software, 2 ed.New York: John Wiley & Sons, 1999.

[28] M. A. Johnson, "Effective and Appropriate Use of Controlled Experimentation in Software Development Research," in Computer Science. Portland: Portland State University, 1996.

[29] E. Simmons, "Defect Arrival Modeling Using the Weibull Distribution," presented at International Software Quality Week, San Francisco, CA, 2002.

[30] C. Kaner, W. P. Bond, and P. J. McGee, "High Volume Test Automation (Keynote Address)," presented at International Conference for Software Testing Analysis & Review (STAR East), Orlando, FL, 2004.

[31] Kaner, "What is a Good Test Case?," presented at International Conference for Software Testing Analysis & Review (STAR East), Orlando, FL, 2003.