International Journal of Trend in Scientific Research and Development (IJTSRD)
International Open Access Journal
ISSN No: 2456 - 6470 | www.ijtsrd.com | Volume - 2 | Issue – 1

Novel Metrics in Software Industry

Dinesh Kumar Y
Assistant Professor, Department of CSE,
DIET, Visakhapatnam

Nuka Raju Kolli
Associate Professor, Department of CSE,
DIET, Visakhapatnam

ABSTRACT

The role of metrics in software quality is well recognized. However, software metrics are yet to be standardized and integrated into development practices across software industry. While process, project, and product metrics share a common goal of contributing to software quality and reliability, utilization of metrics has been at minimum. This work is an effort to bring more attention to software metrics. It examines the practices of metrics in software industry and the experiences of some organizations that have developed, promoted, and utilized variety of software metrics. As various types of metrics are being developed and used, these experiences show evidence of benefits and improvements in quality and reliability.

Keywords: Software metrics, Cost of defects, State of metrics, Metrics in software industry.

1. Introduction

It is yet to be widely recognized that metrics are a valuable treasure an organization could have. They provide measurement about schedule, work effort, and product size among many other indicators. The more they are utilized, the more effective and productive the organization becomes. They also provide better control over projects, and better reputation of the organization and its business practices. Software metrics are utilized during the entire software life cycle. Gathered data is analyzed and evaluated by the project managers and software developers. The practice of metrics involves Measures, Metrics, and Indicators. A *Measure* is a way to appraise or determine by comparing to a standard or unit of measurement, such as the extent, dimensions, and capacity, as data points. The act or process of measuring is referred to as Measurement. While a *Metric* is a quantitative measure of the degree to which a component, system, or process posses a given characteristic or an attribute; an *Indicator* represents useful information about processes and process improvement activities that result from applying metrics, thus, describing areas of improvement.

Instituting a metrics program is a challenge for many organizations, mostly the commitment to upfront investment in gathering data necessary for building useful metrics. In addition to time, cost, and resource factors, developers are often reluctant to collect and archive project data. A commonly cited reason is the misuse of project data against developers and project stakeholders. Team leaders and managers play an important role in the adoption of measurement programs as integral part of the software engineer culture. They need to be convinced of (and committed to) software measurement, and at the same time promote this culture and reward their teams for it.

The area of software measurement has been highly active for several decades. As a result, there are many commercial metrics available in the market. Such (affordable) metrics can be the starting point for small organizations. However, much more work is needed to standardize, validate, and integrate metrics into software practices. This work brings needed attention to software metrics and examines the current state of metrics in software industry. The discussion is motivated by cost of defects and description of commonly used metrics.

2. Cost of Defects

To present a convincing argument for the benefit of using metrics, one needs to highlight the incentives and payoff. Here we refer to an article, authored by William

T. Ward [1], describing Hewlett-Packard's (HP's) "*10 x software quality improvement*" initiative. The author uses data from software metrics database and an industry profit-loss model to develop a method to compute the actual cost of software defects. The database is an important element of HP's software quality activities and is a valuable source for different tasks such as quality status reporting, resource planning, scheduling, and calculation of cost defects. Sources of data include product comparisons, analysis of source code size and complexity, defect logging, project post-mortem studies, and project schedule and resource plans. The Software Quality Engineering Group follows definite steps to discover, correct, and retest a defect during testing activities (integration, system, and/or acceptance). The estimated effort here is about 20 hours, and it represents the average effort for discovering and fixing a defect. This effort is calculated using data points from multiple projects that were tracked with the software quality database. Defect cost can also be determined per project or phase, and cost can be weighted based on programmer productivity or product code size. For instance, the following formula shows the cost per defect that is discovered and fixed during the integration through the release phases of a project.

Software Development Cost = SDRC + PL

Where

SDRC (Software Defect Rework Cost) is determined by the amount of effort and expense required to find and fix defects during the integration through release phases, and PL (Profit Loss) is the revenue loss caused by lower product sales throughout the entire post release lifetime.

To illustrate, a product has about 110 software defects found and fixed during testing. Each defect requires 20 engineering hours to identify and fix. The total work effort is 2200 hours. At $75/hour, SDRC is

$165,000, and the rework cost per defect is $1500. These expenses can be saved had metrics been used to mitigate those defects. In addition, it should be noted that the other calculation for defect cost is product profit-loss. Here, missed market-window opportunities result in loss of sales, profits, and competitiveness. This illustrates typical losses that result from the lack of metrics utilization.

3. Common Software Metrics

Unlike software engineering, other disciplines capitalize on the power of quantitative methods to measure their processes and activities. Based on Tom DeMarco [2] statement, "*You can't control what you can't measure*", these disciplines apply measurements to gain better control of their projects and quality of products. Although software engineering is new and evolving discipline, experts have proposed quantitative methods applicable to all aspects of software projects with the goal of achieving high quality products. These methods are related to different activities including:

Cost and effort estimation: Estimation models [3] help better plan and execute software projects. One factor that plays into the success of applying estimation models is the experience of the organization to predict effort and cost for new software systems. Mathematical models, such Boehm's COCOMO [4], Putnam's SLIM [5], and Albrecht's Function Points [6], can be used.

Productivity measures: Productivity models focus on the human side of the project. A key factor to accurate determination of productivity is having sufficient information about the productivity of an individual (or the team) in different scenarios, such as the type of project, team structure, skills and backgrounds, tools, and environment. Measures and metrics for assessing the human side of the project are more challenging to develop and apply than other measures and metrics [7].

Data collection: An important discipline, requiring diligence and careful implementation. Although it has obvious benefits for developing measures and metrics, team members often dislike it. The common perception among some team members is that data collection leads to uneasy feeling of being "under pressure" and "at risk" as collected data can be negatively used in performance evaluations. The real risk here is that inaccurate data can result in metrics that provide false assessments.

Quality assessment: This activity covers different measures including efficiency, reliability, flexibility, portability, usability, correctness, and many others. Standards that define quality means in terms of specific project goals are needed. Here and with historical data, objectives (in terms of measures) should be achieved or exceeded to meet desired quality standards. Although quality assessment is often applied during early in the life cycle, it covers, along with "umbrella activities", the entire life cycle [7].

Reliability models: Even though reliability is seen as a quality attribute, reliability assessment models are more

related to software failures, and are mostly applied during testing. The models work well when it is possible to monitor and trace failures during a test or operation. Many quality models use reliability as a factor, and the concept of reliability weights much in the perception of quality.

Other activities include: Performance evaluation for optimal solutions, Structural and complexity, Capability maturity assessment, Management by metrics, and Evaluation of methods and tools. These activities are becoming an important part of Software Engineering as each activity leads to the development of software metrics, which some of them evolve into assessment models.

Process, Project, and Product are three common categories for software metrics. Below, we highlight the key focus on each category.

Process Metrics: These metrics focus on software development and maintenance. They are used to assess people's productivity (called private metrics), productivity of the entire organization (called public metrics), and software process improvement. Process assessment is achieved by measuring specific attributes of the process, developing a set of metrics based on the identified attributes, and finally using the metrics to provide indicators that lead to the development of process improvement strategies. Private metrics are designed to help individual team members in self-assessment allowing an individual to track work tasks and evaluate self-productivity. Pubic metrics, on the other hand, help evaluate the organization (or a team) as a whole, allowing teams to track their work and evaluate performance and productivity of the process. A good example is team's effectiveness in eliminating defects through development, detecting defects through testing, and improving response time for fixes.

Project Metrics: Project metrics are tactical and related to project characteristics and execution. They often contribute to the development of process metrics. The indicators derived from project metrics are utilized

by project managers and software developers to adjust project workflow and technical activities. The first application of process metrics often occurs during cost and effort estimation activity. Metrics collected from past projects are used as basis from which effort and time estimates are made for new projects. During the project, measured efforts and expended time are compared to original estimates to help track how accurate the project estimates were. When the technical work starts, other project metrics begin to have significance for different measures, such as production rates in terms of models created, review hours, function points, and delivered source code lines. Common software project metrics include:

- Order of growth: Simple characterization of an algorithm‘s efficiency allowing to compare relative performance of alternative algorithms without being focused on the implementation details.
- Lines of code: The Physical type is a count of lines including comment and blank lines (not to exceed the 25% of all lines of code). The logical type counts the number of "statements" tied to a specific programming language.
- Cyclomatic complexity: Measures the application complexity and describes its flow of control.
- Function points: Reflects functionalities relevant to (and recognized) by the end user. It is independent of implementation technology.
- Code coverage: Determines statements in a body of code that have been executed through a test run and those statements that have not [8].

Other project metrics include coupling, cohesion, requirements size, application size, cost, schedule, productivity, and the number of software developers.

Product Metrics: These metrics focus on measuring key characteristics of the software product. There are many product metrics applicable to analysis, design, coding, and testing. Commonly used product metrics include:

- Specification quality metrics: These metrics provide indication of the level of specificity and completeness of requirements.
- System size metrics: They measure the system size based on information available during the requirements analysis phase.
- Architectural metrics: These metrics provide an assessment of the quality of the architectural design of the system.
- Length metrics: They measure the system size based on lines of code during implementation phase.
- Complexity metrics: They measure the complexity of developed source code.
- Testing effectiveness metrics: They measure the

effectiveness of conducted tests and test cases.

Other product metrics focus on design features, quality attributes, code complexity, maintainability, performance characteristics, code testability, and others.

4. Metrics in Software Industry

Software measurement started in the early 1970s in the US and Canada. The SEI at Carnegie Mellon University helped establish many measurement programs giving a platform to help increase the use of software metrics in software industry. Organizations such as HP, Motorola, NASA, Boeing, AT&T, and others use software metrics extensively [11]. In Germany, since late 1980s companies like Siemens, Bosch, Alcatel, BMW, and others started integrating software measurement programs into their practices. To present snapshot of the state of metrics in software industry, examples from HP, Motorola, NASA, and Boeing are presented in the section to highlight the initial steps and effort toward integrating the practice of measurement in software development. Many consider these initiatives and efforts a significant contribution toward promoting the practice of software metrics.

Hewlett-Packard (HP)

HP's experience in incorporating a software metric program has been one of the most reported initiatives in the industry. Grady and Caswell [9] implemented the program in an effort to improve software project management, team productivity, and software quality. These goals were achieved in the short term for individual development projects. Grady and Caswell categorized metrics into primitive or computed. Primitive metrics are those directly observed such as total development time for the project, number of defects in unit testing, lines of code - the program size and so forth. Computed metrics cannot be directly observed, they are mathematical aggregations of two or more primitive metrics. Examples of most widely used computed metrics at HP include:

- Metrics for project scheduling cost of defects, workload, and project control. For example:
  - Average fixed defects/working day
  - Average engineering hours/fixed defect

Average reported defects/working day

  - Defects/testing time
  - Percent overtime: Average overtime per week
  - Phase: engineering months/total engineering months

- End product quality metrics: For example:
  - Defects/KNCSS (Thousand Non-Comment Source Statements).
  - Defects/Lines of Documentation (LOD) not included in the program source code.

- Testing effectiveness metrics: Example indicator is Defects/testing time.

- Testing coverage metrics: Example indicator is Branches covered/total branches. This indicates what percentage of the decision points in the program was actually executed.

- Useable functions metrics: Example indicator is Bang, which is "a quantitative indicator of net usable functions from the user's point of view" [2]. Bang is computed in two ways: For function-strong systems, computing Bang is counting the tokens entering and leaving the function multiplied by the weight of the function. For data-strong systems, computing Bang involves counting the objects in the database weighted by the number of relationships of which it is member.

- Productivity metrics: Example indicator is NCSS/engineering month.

HP's software metrics program served as a model for many organizations and prompted a wide interest among organizations seeking to improve the quality of their products and software development processes.

NASA

NASA implements software metrics with emphasis on improving reliability in software requirements specification and source code. For complete requirement coverage, test plans are also examined without excessive testing and without expending expenses. To improve reliability, they consider three life cycle phases: requirements, coding, and testing. Software metrics and error prevention techniques can be applied throughout these phases to help improve reliability [12].

Requirements Metrics: For reliability, NASA's metric tool (called ARM - Automated Requirements Measurement) parses requirements document file line by line searching for certain words and phrases. This tool has been used in 56 requirement documents. The

developed measures include [12]:

- Lines of Text: Physical lines a measure of size.
- Directives: References to figures, tables, or notes.
- Continuances: Phrases that follow an imperative and introduce the specification of requirements at a lower level, for a supplemental requirement count.
- Imperatives: Words and phrases that command that something must be done or provided. The number of imperatives is used as base requirements count.
- Options: Words that seem to give latitude in satisfying the specifications but can be ambiguous.
- Weak Phrases: Clauses that may cause uncertainty and leave room for multiple interpretations.
- Incomplete: Statements that have TBD (To be Determined) or TBS (To Be Supplied) clauses.

The ARM software does not evaluate whether the requirements are correct or not, but evaluates the vocabulary and the individual specification of statements used to state the requirements. ARM also evaluates the structure of the requirements document. It identifies number of requirements at each level of the hierarchical numbering structure. This information helps indicate potential lack of structure that may impact software reliability by increasing the difficulty to make changes. It may also indicate unsuitable levels of details that may constrain software design.

Design and Code Reliability Metrics: For design and code reliability, NASA's Software Assurance Technology Center (SATC) developed a tool that analyzes source code for architecture features and structure and to help locate error-prone modules based on source code complexity, size, and modularity. Although there are different complexity measurements, SATC uses Cyclomatic complexity (number of independent test paths). They found that combining size and complexity makes the most effective evaluation. Large modules with high complexity tend to have the lowest reliability. Such modules are reliability risk because they are difficult to change or modify. SATC uses the following metrics for object-oriented quality analysis:

- Weighted Methods per Class (WMC)
- Response For a Class (RFC)
- Coupling Between Objects (CBO)
- Depth In Tree (DIT)
- Number Of Children (NOC)

These metrics lack industry guidelines, and therefore, SATC developed guidelines based on NASA's data and are made available on NASA's SATC website (http://satc.gsfc.nasa.gov/).

Testing Reliability Metrics: For testing, SATC developed a simulation model for error discovery and for projecting number of remaining errors in the source code and when such errors will be discovered. This model is based on the Musa Model. The Musa model, known as the "Execution Time Model", is used to evaluate computer resources with respect to (1) reduction in the number of faults in a computer program, (2) estimation of testing time necessary to find and correct system errors to achieve an acceptable level of errors in the code, and (3) determination of software reliability based on the specified program operating cycle and mean time to fault. Effective verification aims to ensure that every requirement is being tested. In order to make sure that the system has the functionality specified, test cases are developed (based on one system state) to test selected sets of functions that are based on related sets of requirements. Here, the requirement's functionality is included in the delivered system when the test is successful. Assessment of traceability of requirements to test cases is also performed, and therefore, each requirement is tested at least once. Note that some requirements are tested more than once since they are involved in multiple system states.

In addition to reliability metrics applied throughout the lifecycle, NASA developed IV&V Metrics Data Program to gather, verify, sort out, store, and distribute software metrics data. Collected data include metrics and their associated problem and product data, allowing users to explore the correlation between metrics and the software. NASA's metrics include: McCabe Software metrics, Line of Code metrics, Requirement metrics, Error metrics, and Halstead metrics. In an effort to promote metrics utilization in the software industry, NASA offers project non-specific data available in its repository to the software community through the Metrics Data Program website (http://mdp.ivv.nasa.gov/).

Boeing

Boeing's 777 program earned the company recognition for achievements through its metrics program, among

other related software development initiatives [13]. The Boeing 777 program is one of the most software intensive commercial airplanes. It has near 2.5 million newly developed lines of on-board code. Other estimates indicate around 4 million additional lines of code for customer options. The software has over one hundred components corresponding to physical boxes in the airplane's control system. Many of them were produced by third- party companies.

At the beginning of the program various software measures were used by the suppliers and their counterparts to present the status of the work. There was a variety of measures that were hard to understand. About half way through the 777 development program, a uniform use of software metrics was instituted. Suppliers were asked to report simple, standard software metrics including test definition, resource utilization, plans for software design, coding, and test execution. In addition, actuals were collected for software problem report totals.

Boeing's implementation of the metrics program is defined as follows: "*Each supplier was requested to prepare plans for their design, code, and test activities. These plans showed expected totals and the planned completion status for each of the biweekly reporting periods until the task is complete*". Following that, biweekly updates that show the actual development status in terms of completed design, code, and tests are requested. Changes to the estimated total size of the effort are also reported along with plans to reflect new totals. Information from the metrics is shared with the system developers for improvement purposes. The overall metrics program helped Boeing to improve communications with supplier, adjust project plans in conjunction with actual progress, and keep the project on schedule. Key characteristics of Boeing's metrics program, that were instrumental in supporting this process, include uniformity, frequent updates, clear definition, objective measures, and re-planning, which was very encouraged. In addition, Boeing's effort to define measures resulted in a 21-page set of instructions on how to prepare metrics data. The two critical features of the metrics plans were re-planning when needed and past data was never changed.

Boeing's experience shows that metrics were invaluable as they helped in indicting soon enough where program risk points are, allowing early corrective actions. Early on, uniform metrics encouraged application of reasonable checks on plans and discussion of such checks. As a result, communications with suppliers that were prompted by metric data were as important as the metric data itself. In addition, development metrics were used to track progress against the plans for design, code, and testing. They included software size and number of tests. Milestones were indicated on metrics charts, associated with the milestones are success criteria based on design, code, and test completion.

5. Conclusion

The practice of software measurement is lacking behind and yet to mature enough to be prompted widely across software industry. Although the importance of metrics was realized since early 1970's, it is taking a slow pace into the practice of software development. Organizations, such as HP, Motorola, NASA, Boeing, and other organizations, have been developing and applying software metrics to their projects. Although their metrics and those applied by others are based on standards, some organizations tend to adapt these metrics to their process and needs. This was clear in NASA's case were they detected a lack of aids to assist in evaluating the quality of requirements or individual specification statements.

Experiences discussed in this paper and many others cases indicate when metrics are used early in the development cycle they help detect and correct requirement faults and prevent errors later in the life cycle. Software metrics can be used at each phase of the development as illustrated in section 4. Metrics can identify potential problems that may lead to errors in the system. Finding these potential problems decreases over all development cost and prevents side effects that may result from making changes later in the development cycle. Using software metrics need not be time consuming effort. Measurement activities such daily tracking should be developed as a habit rather than burden on developers and the organization

Effort made by the above discussed organizations (and many others) is a step in the right direction for metrics and software development. Being the main goal of this article, dissemination and awareness of such effort and the availability of much of such metrics to the software community (often for free) is very much needed to help forward the evolution of such initiatives and to broaden the utilization of metrics.

REFERENCES

1. http://findarticles.com/p/articles/mi_m0HPJ/is_n4_v42/ai_11400873, Hewlett-Packard Journal, October 2012
2. DeMarco, Tom. *Controlling Software Projects: Management, Measurement & Estimation*, Yourdon Press, New York, USA, 2012.
3. http://sern.ucalgary.ca/courses/seng/621/W98/johnsonk/cost.htm
4. http://sunset.usc.edu/research/COCOMOII/index.html
5. http://www.qsm.com/
6. Albrecht, A. J. and J.R. Gaffney, *Software function, source lines of code, and development effort prediction: a software science validation*, IEEE Trans. on Software Engineering, 9(6), pp 639-648, 2013.
7. Pressman, R.S. *Software Engineering: A Practitioner's Approach.* 6th Edition, McGraw Hill Publishing Company, 2015.
8. http://www.cenqua.com/clover/doc/coverage/intro.html
9. Grady, R. B. and D. R. Caswell. Software Metrics: 13 Quantitative Analysis of English Prose Establishing a Company-Wide Program. Engle- 14 Application to Hardware wood Cliffs, N. J.: Prentice-Hall, 2015