Improving Software Reliability, Productivity and Quality Using Software Metrics

Harminder Pal Singh Dhami
Dept. Computer Science, OPJS University, Churu, Rajasthan, India

Abstract
The Software engineering is an application of a technical, systematic, scientific approach to the development, function, and maintenance of software; that is, the application of engineering to software. Previously software measurements are limited to measuring individual, product software attributes. In system development, metric is the measurement of a particular characteristic of a program's performance or efficiency and which can help to enhance product reliability. It is used by the software industry to quantify the development, operation & maintenance of software and provide information to support quantitative managerial decision-making during the development lifecycle of the software. Software metrics is a term which is used to describe wide range of activities concerned with measurement in software engineering.
This module introduces the most commonly used software metrics and reviews their use in building models of the software development process. Most software metrics activities are initiated for the purposes of risk analysis and software reliability of some form or another. Yet traditional metrics approaches, such as regression-based models for cost estimation and defects prediction, provide little support for managers wishing to use measurement to analyze and minimize risk. Many traditional approaches are not only insufficient in this respect but also fundamentally inconsistent. Considerable improvements can be achieved by using causal models that require no new metrics.

Keywords
SRE, SDLC, LOC

I. Introduction

A software metric is a standard to evaluate an amount or level to which a software system or process possesses some property. Software metrics is an integral part of the state-of the-practice in software development process. It provides a quantitative aspect for the development and validation of models of the software development process. Software metrics can be used to increase software productivity and quality. Now-a-days customers are specifying software and/or quality metrics coverage as part of their requirements. International standards like ISO 9000 and industry models like the Software Engineering Institute's Capability Maturity Model Integrated include quality measurement. The term software metrics means different things to different people. Software metrics can vary from project cost and effort prediction and modeling, to defect mapping and root cause analysis, to a specific test coverage metric, to computer performance modeling.
The relevance of software metrics to a software development process and to a developed software product is a complex task that requires study and discipline, which conveys knowledge of the status of the process and / or product of software in regards to the goals to achieve stage/phase-based defect removal pattern.
The basic aim of software engineering is to produce high quality efficient software at low cost. With growth in size and complexity of software, management issues began dominating. An optimal design strategy without any compromises e.g. cost & time, for the system does not develop an optimal design. The reason for this is the changes in requirements that may occur in later development cycles. Such changes may cause design decisions taken earlier to be less optimal. Design erosion is inevitable with the current way of developing software. Refined methods only contribute by delaying the moment that a system needs to be withdrawn or retired. These approaches do not address the fundamental problems that cause design erosion and makes system unreliable [4]. Component based design is expected to have a strong impact on the quality of software development: Due to the simplicity, the software development speeds up. The shorter development time results in reduced costs. The extensibility and resolvability of software systems is improved, because components can flexibly be substituted by another component that satisfies the requirements. Software components enhance the reliability of the software, as they are improved, tested and debugged over years [5].
Software Reliability Engineering (SRE) is a practice that helps one develop software that is more reliable, and develop it faster and much cheaper. It is a proven standard and best practice that is generally applicable to systems that include software [3]. Software Reliability Engineering works by quantitatively characterizing and applying two things about the product: i) the projected relative use of its functions and ii) its required major quality characteristics. The major quality characteristics are reliability, availability, delivery date and life-cycle cost. In applying software reliability engineering, one can vary the relative emphasis on these factors.

A. Classification of Software Metrics

Software metrics may be classified as process metrics, product metrics and project metrics.
Process metrics, are measures of the software development process, efficiency and quality of software process, usefulness of defect removal, such as complete development time, type of techniques used, or the average level of experience of the development team. The development team will have to decide which process model they use for their project as per customer requirements. As all models have their own pros and cons. However, now a days the fusion of all these methodologies is incorporated. All the developers look at the low cost, risk, high quality and small cycle of time, so that the productivity and quality of the product can be optimized. There should be a tradeoff between the development time and the quality of the product [11].
Product metrics or it is product measurement through product metrics. It measures the complexity of the software design and software product at any phase of development, from initial requirements to implementation of system. It may measure the size of the final program (source or object code), or the number of pages of documentation created.
Project metrics shows the project characteristics and execution. It is used for tactical purposes rather than strategic purposes. Project metrics enables the project managers to assess present projects, identify potential risks, locate problem areas, and assess the project staff's ability to control the quality of work products.
In addition to the distinction between product and process metrics,

software metrics can be classified in other ways. Normally, speaking objective metrics should always result in like values for a given metric, as measured by different competent observers. For subjective metrics, even experienced observers may calculate different values for a given metric, since their subjective opinion is involved in arriving at the measured value. In case of product metrics, the size of the software product measured in Lines of Code (LOC) is an objective measure, for which any well-versed observer, considering the same definition of LOC, should obtain the same measured value for given software. Subjective product metric's example is the categorization of the software as "organic" "embedded," or "semi-de-tached," as required in the COCOMO cost estimation model [1]. Even though most programs might be easy to categorize, those on the borderline between types might logically be classified in different ways by different educated observers. In case of process metrics, an example of an objective measure is development time, and level of analyst experience is likely to be a subjective measure. An additional way in which metrics can be categorized is as computed or primitive metrics[6]. Computed metrics are those that can be indirectly observed and are computed in some manner from other metrics. Examples of computed metrics are those generally used for productivity, such as LOC created per person-month, or quality of product, such as the number of defects per thousand LOC. Computed metrics are combinations of other metric values and thus are often more important in understanding or assessing the software process than are simple metrics. Primitive metrics are those that can be observed directly, such as the size of program, number of observed defects in unit testing, or total development time for the project.

II. Methodology

There are various steps that provide a practical, orderly method of choosing, designing and implementing software metrics.

A. Identify Metrics Customers

The initial step is to identify the customers for each metric. The metrics customer is the person or people, who will be taking action or making decisions based upon the metric; the people who wants the information supplied by the metric. There are many various types of customers for a metrics program and this may add complexity to the program because each customer/client may have dissimilar information requirements [2]. Customers may include:

- Project Management
- Functional Management
- Software Engineers/Programmers
- Testers/ Testing Expert
- Specialists
- Customers/Users

B. Attainable Goals

The next step is to select assessable goals. The goals we select to use in the Question/Metric will differ depending on the level we are taking into account for our metrics. As at the organizational level, we typically study high-level strategic goals like maintaining a high level of customer satisfaction, being a low cost provider or profit margin target or meeting projected revenue. At project level, we normally consider goals that lay emphasis on project management and control issues or the objectives and requirements of this level. These goals in general reflect the project success factors like on finishing the project within schedule, timely delivery, completing the project within budget or delivering software with the required level of quality, attaining the essential performance level.

C. Define Questions

This step is to identify the questions that are required to be answered in order to ensure that each goal is being achieved. For example, if our goal was to deliver only defect-free software, we might select the questions:

- Is the software product effectively tested?
- Are all known deficiencies corrected?
- Is the software product efficiently running?
- Does any defect still exists or remains undetected?

D. Metric Selection

In this step metrics selection is done that provide the information needed to answer the above questions. By now each selected metric has a clear and specific objective, to answer questions that need to be answered in order to determine are we are meeting our goals? Metrics can help us to –

- Understand more clearly about our software products, processes, controls and services.
- Evaluate our products, processes, controls/checks and services against established standards and goals.
- provide the information we need to Control resources and processes used to produce our software product.
- Predict attributes of software entities in future.

E. Standardize Definitions

This step is to agree to standard definitions for the entities and their measured attributes. When we use terms like defect, fault, problem report, size, failure and even project, some people will understand or interpret these keywords in their own context with meanings that may vary from our intended definition. These understanding or interpretation differences increase when more ambiguous terms like quality, maintainability, reliability and user-friendliness are used. Additionally, individuals may use different terms to mean the same thing. For example, the terms defect, fault, report, problem report, failure, incident report, fault report, or customer call report may be used by various institutes or business houses to mean the same thing, but unfortunately they may also refer to different entities.

F. Select a Measurement Function

This is to select a measurement function for the metric. In simple terms, the measurement function describes how we are going to determine the metric. Some metrics, called base measures or metric primitives, are measured directly and their measurement function normally consists of a single variable. Examples of base measures include the number of LOC (Lines Of Code) reviewed during an assessment or the hours spent in preparing for an assessment meeting. Other more complex metrics, called derived measure is modeled using mathematical combinations of base measures or other derived measures. An example of a derived measure would be the evaluation's preparation rate, modeled as the number of LOC reviewed divided by the number of preparation hours.

G. Define a Measurement Method

This step in designing a metric is to decompose the function into its lowest level base measures and identify the measurement method used to give/input a value to those base measures. The measurement method defines the mapping system that is used to assign values or symbols to the attributes. In order to implement the metric, the base measures and their counting technique

define the initial level of data that needs to be collected. Some measurement methods are implemented by using standardized units of measure whereas other measurement methods are based on counting criteria, which are regular count of objects with certain characteristics. For example, if the metric is the error report event rate per month, we could merely count all of the error reports in the database that had a release or start date during each month. However, if we wanted problem or defect counts instead of just error report counts we might rule out all the reports that didn't result from a product defect. Other rules may also be introduced or used for the measurement method.

H. Identify Decision Criteria

In designing a metric, identification and defining of decision criteria is done in this step. Once we have decided what to measure and how to measure it, we have to decide what to do with the outcome. According to the ISO/IEC 15939 Software Engineering - Software Measurement Process Standard, decision criteria are the "thresholds, targets, or patterns used to determine the need for action or further investigation or to describe the level of confidence in a given conclusion or result".

I. Specify Define Reporting Mechanisms

This step is to specify how to report the metric. It includes defining the report format, data mining, data reporting, distribution cycle, reporting mechanisms, and availability. The report format defines- what the report looks like? Is the metric included in a tabular form with other metrics values for the period? Is the latest value in a trend chart that tracks values for the metric over various periods added? The trend chart be a bar, line, or area graph? or Is it better to compare values using stacked bars or a pie chart? Do the reports specify tables and graphs as stand alone, or is there any analysis details included? Are objective or control values included in the report? The reporting mechanism summarizes the way that the metric is delivered.

J. Determine Additional Qualifiers

This second last step in designing a metric is used in determining the additional metric qualifiers. A better metric is a generic metric. It means that the metric is applicable for an entire hierarchy of supplementary qualifiers. For example, we can talk about the duration of unplanned activities for an entire product line, an individual product, or a specific release of that product. We could look at activities of customer or business segment. Alternatively, we could look at them by cause or type.

K. Collect Data

The question of "what data to collect?" was actually answered in above steps vii and x. The objective is to collect all of the data required to provide the metrics primitives and the additional qualifiers. In most cases, the "owner" of the data is the best answer to the question of "who should collect the data?" The data "owner" is the user with direct access to the source of the data and in many cases is actually responsible for generating the data.

III. Result

Software metrics research and practice has helped to build up an empirical basis for software engineering product. This is a significant achievement, but the classical statistical-based methodologies do not provide managers with decision support for anticipating and evaluating risk and increasing the reliability of the software product.

As reliability prediction activity finds future software reliability based upon existing software metrics and measures. It analyzes previous data and findings. Depending on the software development level or stage, forecast involves different techniques. Using reliability prediction models, software reliability can be estimated early in the software development cycle and enhancements can be initiated to improve it.

Hence, it does not satisfy one of the most important goals of software metrics. The immense challenge for the next few years is to use software metrics in such a way that they address the stated key objective.

IV. Conclusion

A metrics program that is based on the objectives of an organization will help communicate, measure progress towards, and ultimately attain those defined objectives. People will work to achieve what they believe to be important. Well-designed metrics with documented purposes can help an organization obtain the information it needs to continue to improve its software product's reliability, processes, and services while maintaining a focus on what is important. A practical, logical and systematic, start-to-end method of selecting, designing, and implementing software metrics is a valuable aid.

References

[1] V. R. Basili, H. D. Rombach,"The TAME Project: Towards Improvement-Oriented Software Environments", In IEEE Transactions in Software Engineering 14(6) (November), 1988.

[2] Norman E. Fenton,"Software Metrics, A Rigorous Approach", Chapman & Hall, London, 1991.

[3] J. Musa, A. Iannino, K. Okumoto,"Software Reliability: Measurement, Prediction, Application", McGraw-Hill, 1990.

[4] H. P. S. Dhami, Dr Anuj Kumar,"Analysis of Software Design Erosion Issues", International Journal of Advanced Research in Computer Science and Software Engineering 3 (7), pp. 1-7, 2013.

[5] A. Kaur, H. P. S. Dhami, J. Kaur (2009),"Component Based Software Engineering", IEEE-SOFT-3140, IEEE International Advance Computing Conference IACC-2009.

[6] Robert B. Grady,"Practical Software Metrics for Project Management and Process Improvement", Prentice-Hall, Englewood Cliffs, 1992.

[7] Paul Goodman,"Practical Implementation of Software Metrics", McGraw Hill, London, 1993.

[8] Capers Jones,"Programming Productivity", McGraw Hill, New York, 1986.

[9] Watts Humphrey,"Managing the Software Process", Addison-Wesley, Reading, 1989.

[10] IEEE Standard Glossary of Software Engineering Terminology, ANSI/IEEE Std 610 (1990), The Institute of Electrical and Electronics Engineers, New York, NY, 1990. [ISO/IEC-15939] ISO/IEC 15939:2002 (E), International Standard, Software Engineering – Software Measurement Process.

[11] H. P. S. Dhami (Sept.2016),"Comparative Study and Analysis of Software Process Models on Various Merits", International Journal of Advanced Research in Computer Science and Software Engineering, Vol. 6(9), pp. 234-243.

H. P. S. Dhami received his B.E. degree in Computer Science from Poona University, India, in 1994, the Masters degree from Punjabi University, Patiala, India, in 2003, and pursuing the Ph.D. degree in Computer Science. He has been serving the academics & I.T Industry (of National & International repute) for more than the last 22 years. He has served at various positions as Faculty, Head of Department, Technical Advisor, Project Guide, Senior Manager, Exam Manager (for various National & International Universities), and Member Management of various educational organizations. He has authored two books and more than 15 research papers in National and International Conferences/ Journals. His research interests include System Analysis and Design, Software Engineering and Software Testing.