Software Engineering Process Measurement Report

Name: Qichen An

Student Number: 19324317

03/11/2019

Table of Contents

Introduction

The concept of software engineering was first proposed at the NATO Conference, which took place in Garmisch in October 1968. Since then, software engineering has become a traditional branch of engineering subject based on theoretical foundations and practical application. Over the past years, software engineering measurement has emerged as a successful research direction, especially in the direction of mining software engineering data (1). Along with this progress, the efficiency of software development and the quality of the delivered products has increased gradually to be more appropriate to meet requirements. The quality and efficiency are much more significant at present (2). In the area of software development, measurement plays a critical role in providing scientific basis to make software engineering perform effectively and efficiently. Therefore, how to measure and assess software engineering process has been a crucial subject in the present world and this report will focus on the ways of software engineering process measurement.

The rest of the paper unfolds as follows. The next section is devoted to assessment method and basis including 5 parts: measurable data, computational platforms, algorithmic approaches, ethics and some other basis. These factors will continue to affect software engineering for many years. Section 3 is the conclusion and in section 4, we show the reference list.

Assessment Method

Measurable Data

Measurable data may be the most obvious to analyze software engineering processes. However, no prescriptive way can combine or evaluate the meaning behind Software engineering process and code. So that good engineers need to measure correlative variables of a process to comprehend and ameliorate it. There are two types of measures, product metrics and process metrics, in engineering discipline. The former of product metrics is about product's qualities, where dimensions and physical data are usually measured, specifically, in software engineering, maintainability, code length, reusability, complexity included. The latter of process metrics, which assess the quality according to engineering process such as production time and effort required, includes number and type of changes, editing time in software engineering (3).

The availability of measurable data**,** from software repositories which provides different kinds of information to help enhance understanding of the software, prompts developers to create varieties of models to assess the quality of a software process and products. (4). Measures represent data, which enables developers to understand how things work and how to improve the software and achieve the expected results, in all engineering disciplines.

Nevertheless, collecting measures of good utility in software engineering is difficult while some developers do not realize importance it these measures. Moreover, Current reference models for software engineering process assessment are to some extent unreliable, as they are primarily built on manual acquisition of evidence of practices, probably leading to unconscious errors. However, external audits (CMMI, ISO 15504, and ISO 9001) which utilized by common reference model assessment standards, are still performed manually to gather compliance evidence instead of automated tools, for the reason that many factors could influence ratings in real world process execution (5).

Another reason, for measuring a developer’s productivity being a tough puzzle to solve, is that there are no objective metrics that could determine the ratings directly. That means, though using metrics might be the most straightforward and effective way to evaluate the process, the metrics can be deceptive and may not work for every developer.

As follows are some key metrics worth considering.

Source Lines of code (SLOC). This method was created in the early time which the program and code just be created, used to judge each software engineer's contribution to the whole development and to predict the amount of effort that will be required. With the rapid development of programming technology, the disadvantages of this method are gradually discovered. For example, a program with more efficient algorithm or low readability would contain less lines of code, which may produce different result of the program rating. Moreover, SLOC can be influenced by some other different factors including programming language used.

The frequency of commits to repositories (6). This software engineering method about version control is favorable to improve the adoption of best practices. Committing frequently to repositories could be a prerequisite for a software product to reverse easily, to select specific version to release and to isolate one change from other changes. Kurt and Leif cited an example that shows a repository on the GitHub website and the author of this repository was unable to tell which changes the commit contains because of his infrequent commits. It indicates that the frequency of commits is closely related to productivity of developers. Furthermore, the frequency of commits shows the activity of software engineers to make contribution to the project.

code coverage. This indicates the degree to the source code of a program that is executed when testcases run. If code coverage is at a high level, a higher percentage of source code can be test and undetected software bugs would occur in a lower possibility.

code churn This is a measure that tells the rate at which the code evolves. Specifically, it is measured as a result of percentage, the lines of code representing an edit (modification, add and delete code) to recent code over short period of time for example a few weeks, divided by the total number of lines of code added. Code churns vary among teams, individuals, different projects and different steps. The reasons about why code churn increase to a high level usually are usually of difficult problem, indecisive stakeholder, unclear requirements or prototyping, polishing for a release. According to code churn, software engineers identify if the process is on the track or if something is wrong.

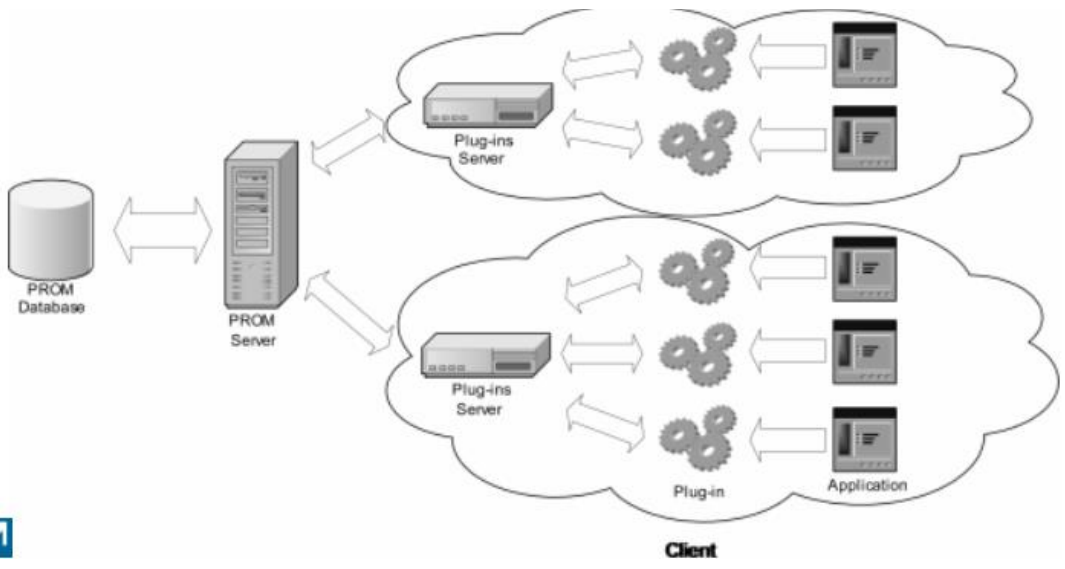
Computational platforms

A computing platform is the stage where computer programs can run and a kind of environment which a piece of software is executed on. It can be considered not only as a constraint but also as an assistance to the development process. With the assistance of computational platforms, engineers can be supported to measure the progress and productivity of a software development.

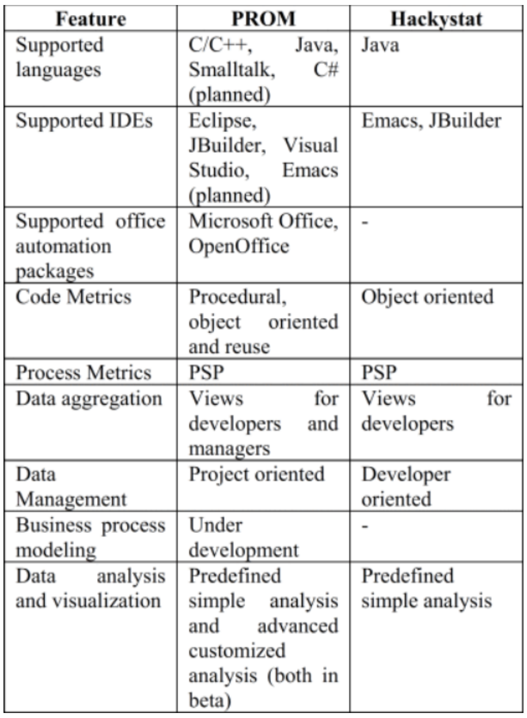
Integrated Development Environments (IDE) (8) Many software developers choose to work in an integrated development environment. Eclipse, a typical integrated development environment, is of great prevalence among Java developers. The common reasons for Eclipse's popularity include abundant Java Development Tools (JDT) support and a plug-in architecture system which allows tight integration of third-party functionality. Murphy, Kersten and Findlater in their report collected usage information from 41 Eclipse users of Java software development, and then deduced users' work habits. Therefore, the conclusion is that the IDEs provide a number of functionalities, and IDEs could be very helpful for projects.

Hackystat (3). Hackystat is a framework which allows individuals to automatically collect and analyze Personal Software Process (PSP) data. This architecture, developed by University of Hawaii, is the third generation of PSP data collection tools. The system uses sensors attached to development tools that use the SOAP protocol to communicate with a centralized server, to collect data and employs. Hackystat can generate metrics, line of codes committed, number of build attempts and failures, code churn and so on included. There are two advantages of using the framework of Hackystat. Firstly, it is very lightweight. The second superiority is that it is entirely based on standard protocols and open-source components, for instance, XML, Apache Tomcat and SOAP. It uses XML files instead of back-end database to store information. In conclusion, this tool places emphasis on individual data collection and the use of individual data.

PROM (PRO Metrics) (3). PROM is a tool which can collect and analyze software metrics and PSP data automatically, which provides different reports to users of different roles inside the project. It is designed to assist developers and managers to keep the projects on the right track. Moreover, PROM is component-based and is based on the Package-Oriented Programming development technique particularly.



And the following table shows Hackystat and PROM comparison.



Feature	PROM	Hackystat
Supported languages	C/C++, Java, Smalltalk, C# (planned)	Java
Supported IDEs	Eclipse, JBuilder, Visual Studio, Emacs (planned)	Emacs, JBuilder
Supported office automation packages	Microsoft Office, OpenOffice	-
Code Metrics	Procedural, object oriented and reuse	Object oriented
Process Metrics	PSP	PSP
Data aggregation	Views for developers and managers	Views for developers
Data Management	Project oriented	Developer oriented
Business process modeling	Under development	-
Data analysis and visualization	Predefined simple analysis and advanced customized analysis (both in beta)	Predefined simple analysis

Pinpoint. This powerful performance measuring tool, launched in 2017, focus on system data and organizational level data. Specifically, it focus on opaque metrics including defect ratio, Backlog change, cycle time and rework rate. In addition, Pinpoint applies data science to provide insights about the development that if the cycle time is trending in the wrong direction the team will receive a diagnostic on where and why.

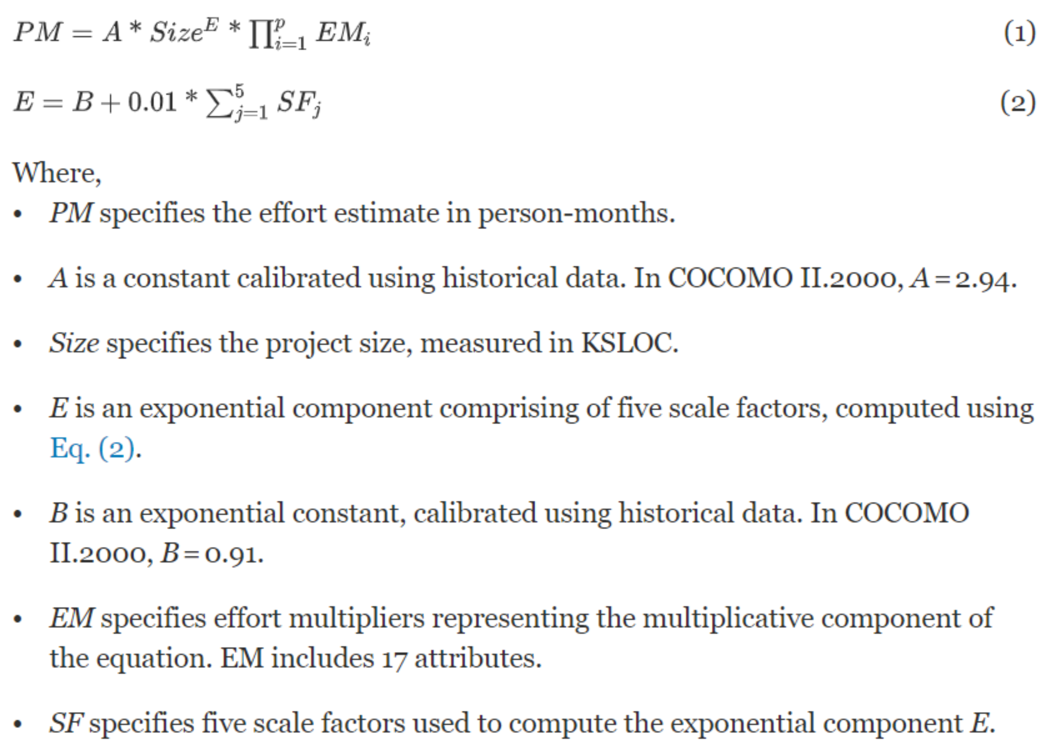
Some other computational platforms also are worthy considering such as GitHub, Codacy, Code climate, GitClear and Gitcolony.

Algorithmic approaches

There are two main approaches to software estimation: model-based and Expert judgment estimation. Combing these two methods is a proposed research direction of software estimation as each approach has advantages and disadvantages according to different contexts. Evidence shows that model-based estimation becomes dominant and there is an algorithmic approach called COCOMO widely investigated which as well as regression-based estimation are the most popular methods currently.

COCOMO (8). algorithmic software development cost estimation model COCOMO is an abbreviation for Cost Construction Model, which is based on LOC. It requires each attribute’s accurate input values to reliably predicting various of parameters such as size, effort, cost, time and quality. The popularity of COCOMO is due to its flexibility; it can be used in different environments and it covers a variety of factors. Originally, Boehm introduced COCOMO (COCOMO 81) as a parametric estimation model. In the early 2000 s at the USC Center for Systems and Software Engineering, COCOMO II, an extended version of COCOMO 81, was published by him and his colleagues. COCOMO II has an exponential component containing calibrated rating values as well as newly five scale factors and constants, contrasted with the first version. What’s more, it has new cost drivers. These indicators are evaluated with pre-defined rating levels from ranges of Very Low to Extra High.

Formula of COCOMO II's effort estimation:



$$PM = A * Size^{E} * \prod_{i=1}^{p} EM_i \quad (1)$$

$$E = B + 0.01 * \sum_{j=1}^{5} SF_j \quad (2)$$

Where,

- *PM* specifies the effort estimate in person-months.
- *A* is a constant calibrated using historical data. In COCOMO II.2000, $A = 2.94$.
- *Size* specifies the project size, measured in KSLOC.
- *E* is an exponential component comprising of five scale factors, computed using Eq. (2).
- *B* is an exponential constant, calibrated using historical data. In COCOMO II.2000, $B = 0.91$.
- *EM* specifies effort multipliers representing the multiplicative component of the equation. EM includes 17 attributes.
- *SF* specifies five scale factors used to compute the exponential component *E*.

Besides COCOMO, another algorithmic approach might be associated to artificial intelligence to analyze the behavior of developer. The concept in the AI area should be introduced such as machine learning, neural networks and fuzzy logic. With AI approaches, the development process can be simulated and even predicted.

Ethics

With the improved approaches to measure the software engineering process, another topic about ethic concerns and relevant problem appear gradually in people's field of vision and become more and more significant. Roy et al. (9) described a novel system to determine the position and location of people in an office environment such as hospitals. Despite the success of the active badge location system installation, most employees first learn about the system are not sure if they would like to use it in their own work situation. They concern their own privacy. For this reason, during the software engineering process, ethic factor should be considered and be evaluated. For instance, for the locator system, it is reasonable to be designed that allows users to specify who have the rights to locate them and also provide information in regard to who and how often someone tried to locate them. Consequently, when the systems are misused, people can realize it. In conclusion, we should try to ensure that applying this technology can bring benefits to employees, rather than menace their privacy.

The rights of individuals must be protected to prevent location systems from being abused. Legislation must be set up to avert misuses of location system, while still using it to its advantages. Such legislation may prevent an employer from being dismissed due to productivity assessment evaluated by location-data logs, as well as compulsive requirement of wearing a locator.

Conclusion

In the report, I have talked about measurable data, computational platforms, and algorithmic approaches. The software engineering is so sophisticated and vague that there is no theory can claim that the assessment is perfect and accurate to measure the quality of a process. However, I believe that we can use multiple data, theories and platforms to get a more objective measurement, as we have seen the development of process measurement over the past few years. We can achieve it now or in the near future, and deal with ethic concern issues at the same time.

Reference List

(1) Hassan, A.E. and T. Xie, Software intelligence: the future of mining software engineering data, in Proceedings of the FSE/SDP workshop on Future of software engineering research2010, ACM: Santa Fe, New Mexico, USA. p. 161-166.

(2) Stephen H. Kan. 2002. Metrics and Models in Software Quality Engineering (2nd ed.). Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

(3) A. Sillitti, A. Janes, G. Succi, and T. Vernazza, “Collecting, integrating and analyzing software metrics and personal software process data,” in Proceedings of the 29th Euromicro Conference. IEEE, 2003, pp. 336– 342.

(4) Di Penta, M. (2012) Mining developers' communication to assess software quality: Promises, challenges, perils. In Emerging Trends in Software Metrics (WETSoM), 2012 3rd International Workshop on. IEEE.

(5) Grambow, G., Oberhauser, R. and Reichert, M. (2013) Automated Software Engineering Process Assessment: Supporting Diverse Models using an Ontology. Int'l Journal on Advances in Software, 6 (1 & 2). pp. 213-224

(6) L. Singer and K. Schneider, "It was a bit of a race: Gamification of version control," Games and Software Engineering (GAS), 2012 2nd International Workshop on, Zurich, 2012, pp. 5-8.

(7) G. C. Murphy, M. Kersten, and L. Findlater, “How are Java software developers using the Elipse IDE?” IEEE Software, vol. 23, no. 4, pp. 76–83, Jul. 2006.

(8) Vu Nguyen, Barry Boehm, LiGuo Huang, Determining relevant training data for effort estimation using Window-based COCOMO calibration, Journal of Systems and Software, Volume 147, 2019, Pages 124-146, ISSN 0164-1212, https://doi.org/10.1016/j.jss.2018.10.019.

(9) R. Want, A. Hopper, a. Veronica Falc and J. Gibbons. The active badge location system. ACM Trans. Inf. Syst., 10(1):91–102, 1992.