CS3012 Software Engineering Measuring Software Engineering Report

Xingchen Yang
17343774

November 25, 2020

Contents

1. Introduction

Software projects always have a reputation for over-schedule and over-budget, and there are also quality problems. The main reason is that there is no accurate forecast of progress, cost and quality, and no effective measurement. The purpose of this paper is to provide a report that considers the software engineering process that can be measured and evaluated based on measurable data, an overview of some analytical platforms that can be used to perform this work, and the ethics surrounding such analysis problem.

2. Measurement and Assessment

2.1. Why should we measure?

As a main tool, measurement can play a role in managing software life cycle activities, assessing the feasibility of project plans, and monitoring whether project activities follow the plan. Software measurement is also the main criterion for evaluating the quality of software products and the ability of organizational software processes. It provides the basis for specifications, management and acceptance specifications for the business agreement between the two parties, and it has become increasingly important.
”Software measurement is a baseline component of good software engineering. You can understand exactly when your development team does their best work and what factors contribute to that.”
Software metrics can provide project managers with various important information about the project, and are also the basis for evaluating project activities. Software measurement can provide basic data for project estimation and planning, as well as quantitative information for controlling the project. At the same time, software measurement also provides instructions for quality management and promotes the process improvement of the enterprise. Metrics provide objective information for software organizations and are a

useful tool to help project managers communicate with projects. Metrics can promote proactive management strategies. Various indicators of measurement indicators, like the red and green indicators of quality goals, help project managers make decisions so that they can weigh cost, schedule, and quality to meet project goals.

Through measurement, the characteristics or attributes of entities can be quantified in order to understand them, predict their future behavior, and ultimately control them.

2.2. What can we measure?

Finding the best individual software metrics to track is extremely challenging. We shouldn't consider data useful just because it is easy to obtain, but then how do we determine which measurements are worthwhile and which ones can give us valuable data? My definition of 'valuable data' is meaningful and useful data, i.e. data which can be analysed to give meaningful results, which are relevant to our goals. To choose the optimal software metrics to measure we should rely on three principles:

• You can effectively measure some area of application development or process.

• A relationship exists between what can be measured and what you want to learn.

• This relationship can be validated and expressed in terms of a formula or a model.

Some of the software metrics which are currently being measured in the industry are:

- Amount of Commits
- Frequency of Commits
- Lines of Written Source Code

• Code Coverage

• Sprint Burndown - complexion of work throughout the sprint based on story points

• Bug Rates - average number of bugs that come up as new features are being deployed

• Mean Time To Repair

• Mean Time Between Failures

• Leadtime - time between the definition of a new feature and its availability to the user

• Cycle time - time between when work is started on an item and its completion.

• Open / close rates - how many production issues are reported and closed within a specific time period.

2.3. How do we assess the data?

The productivity indicators of software development can help we determine which factors hinder the efficiency of the team and eliminate these factors, which ultimately lead to happier.High-performance team. We can compare the costs and benefits of certain practices to determine which cost is worth it. Or we can choose two different benchmarks to choose. A better way. There are many platforms and algorithm methods for analysis. This data will be described in detail in the following sections of the report.

3. Analytical Platforms

3.1. Personal Software Process

Personal Software Process (PSP) is a self-continuous improvement

process that can be used to control, manage and improve personal working methods. It is a structured framework that includes software development forms, guidelines, and procedures. PSP is relatively independent of specific technologies (program design languages, tools, or design methods), and its principles can be applied to almost any software engineering task. PSP can explain the principles of individual software processes; help software engineers make accurate plans; determine the steps that software engineers should take to improve product quality; establish a benchmark for measuring individual software process improvement; determine the impact of process changes on software engineers' ability.

With the popularization of software engineering knowledge, software engineers know that to develop high-quality software, the process of software production must be improved. It is recognized in the industry that SW-CMM, the software capability maturity model developed by CMU/SEI, is currently the best software process, and CMM has become the de facto industry standard for software processes. However, although CMM provides a powerful software process improvement framework, it only tells us "what should be done", but does not tell us "how to do", and does not provide the specific knowledge and skills required to implement key process areas. In order to make up for this deficiency, Humphrey also presided over the development of the Personal Software Process (PSP).

Among the 18 key process areas of the CMM1.1 version, 12 are related to PSP. According to statistics, 70% of the software project development cost depends on the personal skills, experience and work habits of the software developers. Therefore, if a unit’s software developers can receive PSP training, it is a powerful guarantee for the unit’s software capability maturity upgrade. CMM focuses on the macro-management of software processes in software companies and is oriented to software development units, while PSP focuses on the micro-optimization of software processes in companies and is oriented to software developers. The two support and complement each other and are

indispensable.

According to the PSP regulations, the steps to improve the software process first need to clarify the quality objectives, that is, the requirements that the software will meet in function and performance and the potential needs of users. The next step is to measure product quality. It is not enough to have a goal. The goal is only a principled thing, and it is not convenient for actual operation and judgment. Therefore, the goal must be decomposed and measured so that the software quality can be "measured." Then is to understand the current process, find the problem, and adjust the process. Finally, apply the adjusted process, measure the results of practice, compare the results with the target, find out the gaps, analyze the reasons, and continuously improve the software process.

Just as CMM provides a stepped evolutionary framework for the capabilities of software companies, PSP also provides a stepped evolutionary framework for individual capabilities, introducing the concept of the process in a step-by-step manner, and each level contains the lower one. All elements in the level, and new elements have been added. This evolutionary framework is a good way to learn the basic concepts of the PSP process. It gives software personnel measurement and analysis tools to enable them to clearly recognize their performance and potential, so that they can improve their skills and levels.

4. Algorithmic Approaches

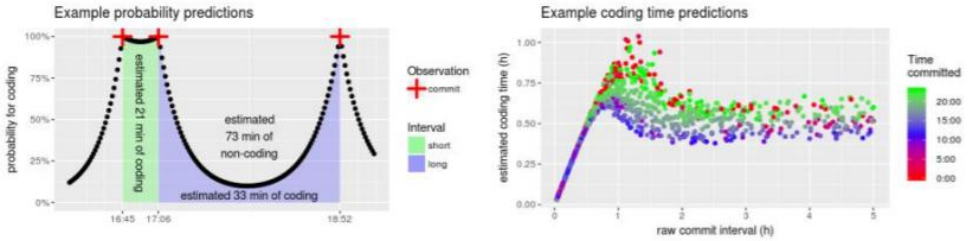
4.1. Neural Hidden Markov Model

We can train a neural network hidden Markov model to predict the possibility of an individual developer currently coding every nearly 1 minute. Therefore, we can compare the expected encoding time between two consecutive submissions with the actual encoding time. There are two states in the model: coded state and non-coded state. We observe a sequence of

submission events with a time stamp-submission can only occur during the encoding process (the probability of training is C per minute). Developers end coding with probability E(t) in the next minute, while non-coding developers start with probability S(t). Compared with the classic Markov model, the probabilities E and S change over time and are affected by trends and habits. Simple Neural Network (NN) with 5 inputs – the sine and cosine of the angle

Fictitious dials on the day and week clocks, as well as the normalized total time – provide probability values. These functions can encode daily or weekly patterns that may change over time. Then, the NN is trained through the back-propagation likelihood gradient of the HMM part.

Neural HMM is a flexible solution to a general problem: for any process that alternates between active and inactive phases, it can change from the observed rhythm of the possibly infrequent "vital signs" (in this case, Submit an event) infer the possibility of being active at any time.). Neural HMM can find night encoders, or developers who only code on weekdays, or weekends, etc. The figure below is a model that maps the submission interval to the coding time estimation.

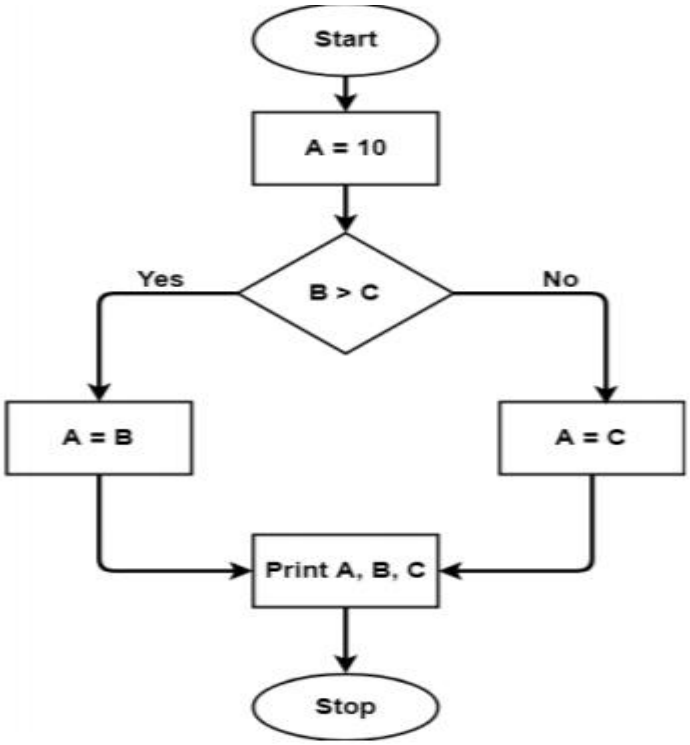


4.2. Cyclomatic Complexity

Cyclomatic Complexity (Cyclomatic Complexity) is a measure of code complexity. It can be used to measure the complexity of a module's decision structure. It can be expressed as the number of independent linear paths in number, and can also be understood as the number of test cases that cover all possible situations and are least used. Large circle complexity means that the judgment logic of the program code is complex, may be of low quality and difficult to test and maintain. The possible errors of the program have a lot to

do with the high cyclomatic complexity.

A sample control flow graph may look like this:



Calculation formula: V(G)=E-N+2P. Among them, E represents the number of edges in the control flow graph, N represents the number of nodes in the control flow graph, and P represents the number of connected components in the graph (the number of components in the graph is the maximum set of connected nodes).

Measuring the complexity of a program is not an easy task, but loop complexity is usually a good way. Why do we bother with the complexity of the code? Well, it is a very valuable metric that can provide a lot of context for other metrics (such as LOC or submission frequency)-a more complex problem may take longer to solve, or in fact a small piece of code actually It is very complicated. In the absence of context, these measurements will be skewed and cannot represent the actual situation. Understanding the data correctly is as important as the measurement itself.

5. Conclusion

The fundamental purpose of software measurement is to help us through

quantitative analysis and summary

Increase productivity, improve product quality, reduce costs and product development cycles. Judging from the successful models of measurement activities in the world, the benefits of measurement activities to organizations and projects are far greater than the costs of measurement activities. With the continuous improvement of software enterprise development process capability maturity level, software measurement will become more and more important; at the same time, with the increase of measurement-based quantitative management components, the predictability of projects will also continue to improve. Software measurement will bring the level of project management to a new level!

6. Source

Top 10 software development metrics
https://www.infopulse.com/blog/top-10-software-development-metrics-to-measure-productivity/
Measuring employee performance
https://resources.workable.com/tutorial/measuring-employee-performance
Nine Software Metrics
https://techbeacon.com/app-dev-testing/9-metrics-can-make-difference-todays-software-development-teams
Personal Software Process
https://en.wikipedia.org/wiki/Personal_software_process
Neural Hidden Markov Model
https://semmle.com/assets/papers/measuring-software-development.pdf
Software Metrics
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.63.2683&rep=rep1&type=pdf
Useful Software Analytics
https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6509376
Algorithmic Approaches
https://pdfs.semanticscholar.org/b8f7/b42d6c693b1abcf112b0b77b504ab89c07bc.pdf