**A report on measuring and assessing the software engineering process**

Ruijie Xiong 19324226

**Abstract**

This report examines the process of measuring the software development process and deploying it to engineers in terms of measurable data, platforms available, algorithmic approach and ethical concerns. Related concepts and theories are introduced, some suggestions to existing problems are provided as well.

**Introduction**

Ever since the idea of software engineering was introduced, Experts have been debating on how to measure and assess the software engineering process. According to Fenton and Pfleeger (1998), the concept of measurement can be defined as “the process by which numbers or symbols are assigned to attributes of entities in the real world in such a way so as to describe them according to clearly defined rules”. In other words, the measurements of the software engineering process can be classified into different attributes of entities, any of which could describe the process in a certain aspect by various rules. There are two main entities in the process of software development, which are developers and the software itself. However, the majority of software engineering metrics and rules were designed for assessments at the organizational level, measuring the software, rather than the individual level, measuring the developers, before the mid-1990s’ (Johnson et al, 2003).

There exists less controversy in the quality measurements of software engineering in the organization level, and some industry standards have been set. As for the individual level, some argue that it is nearly impossible to have a set of measurements to objectively assess developers’ work, while others believe that some of the rules and metrics may vary between different companies, creating subjective assessments to meet the goal. With the big data network environment and the ability to monitor internet activities now, it suggests that the latter opinion would be beneficial to companies and software development in general, it could also be achievable in the near future.

**Measurable Data**

In today’s age of big data, a software engineer can generate a massive amount of data when developing software, the question is which of those data should be considered and measured to assess a developer. In a 2015 campaign by York, over 300 developers were asked on their opinion about the best metrics to assess a software developer’s performance, and the results are shown in figure 1 below.

Looking at the pie chart, speed, personality, lines of code and test coverage all have over 10% votes, but often in situations they cannot accurately reflect the developer’s performance. First, developing speed may vary depending on the tasks, and a high speed does not always lead to high quality code. Then, developers’ personality has little connection to their work performance, as well as hard to be quantified. Furthermore, lines of code and test coverage do reflect performance in certain ways, but may encourage code redundancy, resulting in low quality software. The same could be said to other metrics mentioned by developers. Thus, a single metric tends to only reflect on one aspects of engineers’ work, the idea of “the magic metric” does not exist, due to the complex nature and components of software engineering.

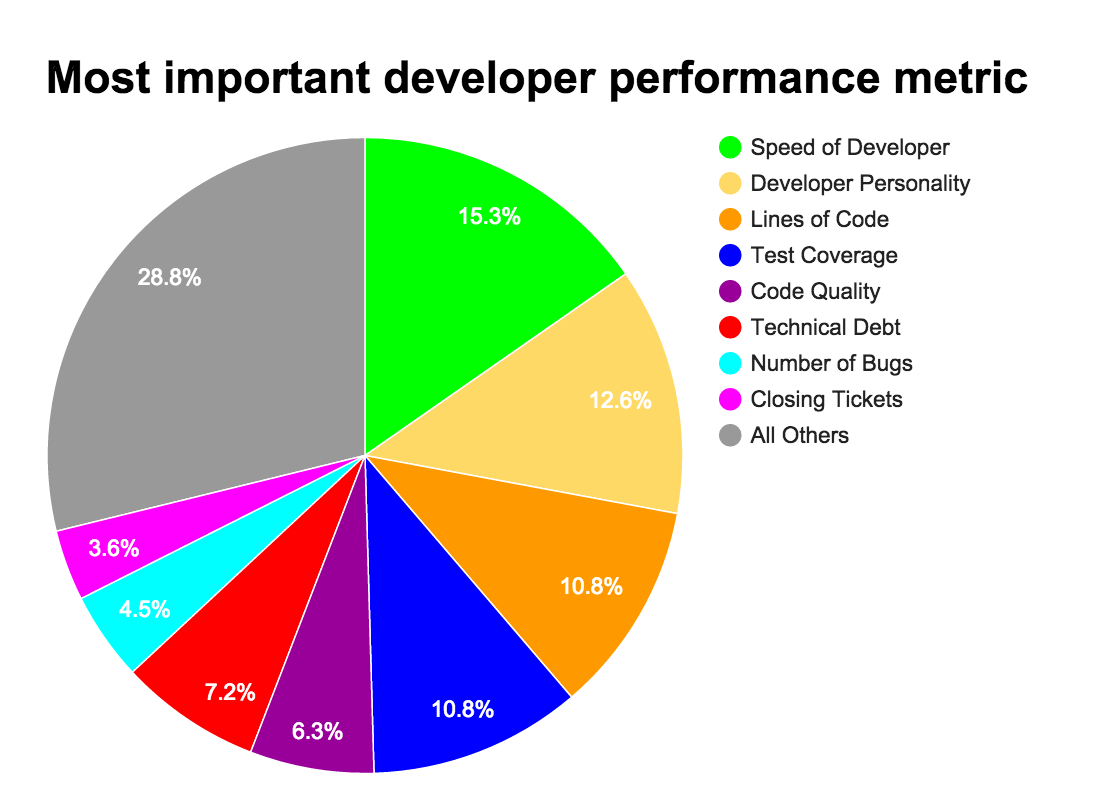


Figure 1. developers’ response on the best metric

Since there are no one or a few metrics that could determine the performance of developers’, the collected data would be in large quantity, reflecting on different aspects of their work, together assessing their performance. There are many components of a software engineer’s work, many of which may be measurable. According to Sandowski and Zimmermann (2019), the majority of work done on developers’ productivity measurements have been using a top-down approach, to assess developers from a manager’s point of view. In other words, these measurements focus on an output-oriented factor of producing high quality codes in a rather short amount of time, without taking other elements in developers’ work into consideration. However, from a developers’ perspective, many other factors such as their roles in the team, tools used and office environment could affect their productivity. Thus, by taking into account developers’ performance on a personal level and an organizational or community level, their work could be measured more accurately.

On a personal level, professional knowledge and skills, productivity and the quality of his/her work are important aspects of assessing if he/she is good at their profession. However, this is not enough for company employees, who often work in teams and under company disciplines to deliver products. To be a good employee, one should be able to fulfill their responsibilities to the company, be actively engaged in their work, as well as make effective communication with teammates.

Many data and metrics can be considered to measure a developer’s personal skills. To test developer’s code base knowledge, lines of code, speed, number of bugs and programming techniques used can be measured. A good knowledge would be considered as producing codes with little errors and warnings, as well as a rather fast speed and using techniques to avoid code redundancy. For examining if the code meets the requirements, defect rate, test coverage, test complexity and test related data could be measured. In addition, integration testing data after the codes have been handed in could contribute to the assessment of requirements satisfaction, by doing further testing in a more realistic environment. A high test coverage rate, as well as successful further testing is a good sign of a piece of desired work. As for the measurement of code quality, the number of tickets closed and the amount of technical debt in the code would be ideal. More tickets closed indicates the developer’s problem solving ability, and a small percentage of technical debt is an important sign of high quality code. A developer’s personal abilities could be measured by various metrics combined in different aspects, together forming a more objective view of developer’s performance.

Modern software developers do not only focus on codes, maintaining effective communications with colleagues and the ability to use tools to achieve goals have become more and more important, especially with the rapid development of Open Source Software (Gousios et al, 2008). To measure a software engineer’s performance on a community level, an online platform where all employees publish work and make discussion should be used, for example GitHub. On GitHub, data on the number of commits, number of pull requests reviewed, as well as the amount of issues created and solved could all contribute and measure the activeness and level of engagement of employees. In addition, peer assessment, for instance how well do other teammates understand and agree with their work, is of significant importance to teamwork. To examine the degree of fulfillment to company responsibilities, information on time took to finish given task and product quality could be collected. Furthermore, some companies expect employees to be learning and improving their skills, then the rate of change of employees’ productivity, the number of tools used and the skill level of using different tools could be recorded for further analysis.

**Platforms available**

Following the discussion in the last part, various metrics need to be taken into consideration when measuring the productivity of software engineers. Putting this notion into action, many software have been developed for managing developers and teams for managers in recent years. GitPrime, one of the most widely used performance measuring tools today, is the first company to commercialize the idea in 2015. Since then, other products including GitClear, Velocity, Pinpoint, and Code Time, a Visual Studio Code extension aim to record programming metrics as well as generate reports and visualization, have been appearing on the market.

Introducing such performance measuring software into the developing process have been significantly beneficial to boosting the development time and quality and decreasing development costs (Harding, 2019). According to a survey conducted by GitPrime on software development productivity, which 1,000 developers from the US and UK participated in, the things that most engineers voted to drain on productivity are not related to technical problems, but on waiting for other people and doing meetings. By making use of these tools, a manager would know when a team is struggling and should intervene in time, helping the developer on a personal level. On an organizational level, such software help managers plan and distribute tasks, track their teams’ progress and make decisions without spending time on traditional meetings and reports, giving developers more time to devote to producing quality product.

GitPrime is the first and one of the largest companies in performance measurement, have clients such as Disney, Adobe, Tesla and Atlassian. GitPrime target non-technical managers, turning the software engineering process and developers’ individual work that are usually difficult to understand into concrete data and analysis to help leaders make decisions. GitPrime records and analyzes data in many metrics, including personal metrics such as number and time of commits, amount of technical debt, and defect rate, as well as organizational metrics including level of activeness in team and progress on current tasks. By looking at the analysis rather than the actual code, managers are able to identify developers who are struggling and need help. In addition, GitPrime not only monitor developer activities on a personal level, but also view the team as a whole, providing services to track the progress of the team and team members, as well as setting development goals. Furthermore, the focus of GitPrime is not to produce evaluation on the code, which could be done by many other tools, but to measure the whole software engineering process (Craven, 2018). It makes the process more transparent and visible to managers and stakeholders. According to GitPrime’s case studies on their clients, most of which have 5% to 15% improvements in their development process, and in some cases extreme improvements such as Storyblocks’ 137% increase in codebase impact.

Another performance measuring software is GitClear, launched in 2017. The biggest difference between GitClear and GitPrime is that GitClear targets managers with a technical background, so GitClear is more focused on codes as well as have a more clear and precise view. GitClear provides managers with a more in-depth examination into the specific code and lines, and a powerful code review tool for more efficient code review process. GitClear calculates different metrics such as code churn, activity type, related commits and ignorable files, which are factors based on a single line, a commit, a file or a branch, resulting in a single metric called “Line Impact” that is used for evaluation. Being a tool that targets technical leaders, it could allow managers to modify the calculation of Line Impact based on their own knowledge or to their preference.

Other popular performance metrics software include Velocity by Code Climate, launched in 2018. Compared to GitPrime, Velocity is more focused on codes, rather than people. For example, when a problem is spotted, Velocity would point out which piece of code is likely to the be problem, while GitPrime would identify the developer who may be struggling. Velocity also have some advance metrics, metrics that are calculated with less advanced metrics, to measure engineers’ work. In addition, Velocity offer a little bit more flexibility than GitPrime. Velocity allow managers to create their own reports with sections that they find more useful. Velocity also offer the option to turn off certain metrics that managers find to be less useful and willing to neglect.

Pinpoint is another powerful performance measuring tool launched in 2017. Differing from the tools discussed above, Pinpoint focus more on system data and organizational level data rather than the actual code written. Furthermore, Pinpoint focus on more opaque metrics such as cycle time and rework rate, rather than straightforward metrics including lines of code and number of commits. In addition, Pinpoint applies data science to provide insights about the development. Pinpoint also has a problem identifying process driven by artificial intelligence. Pinpoint’s AI learns how the team works in real time when collecting data, then modify itself to suit different teams that develop in different ways.

**Algorithmic approaches**

As for the specific algorithms and model to analyze collected data, many researchers have introduced different approaches over the past few decades. Among those methods, the most popular models are considered to be the Constructive Cost Model (COCOMO), Function Point-based Model (FP), Neural Network Model (NN), and Case-based Reasoning Model (CBR) (Park and Beak, 2008). In the area of artificial intelligence, Fuzzy Logic and Natural Language Processing can be used for such analytics as well.

Published in 1981, COCOMO is a model made to assess the effort, cost, and schedule of software development. Generally, COCOMO evaluate software projects by fitting the data of different metrics into a regression formula. COCOMO consists three levels of evaluation with increasing accuracy and detail: basic COCOMO, intermediate COCOMO and detailed COCOMO. In addition, COCOMO also divide software projects into three categories: organic, semi-detached and embedded. For basic COCOMO, only one software metric is used, which is Kilo Lines of Code (KLOC), fitted into the formula with the coefficients corresponding to the project type, to calculate the effort, development time and required persons. This method is for a rough estimation at the beginning stages of software development. Intermediate COCOMO evaluates 15 factors besides KLOC, each factor’s value are multiplied to calculate the Effort Adjustment Factor (EAF), which will be used in the regression formula with KLOC to calculate the final result. Furthermore, in detailed COCOMO, the development process is divided into 6 phases, the effort of each phases are calculated separately before being summed up to form the final result. Thus, intermediate COCOMO and detailed COCOMO both provide more precise estimation, and the idea of using regression models can be applied to the measurement of other software development attributes.

Introduced by Allan J. Albercht in 1979, Function Point Analysis (FPA) analyze a software purely from a user’s point of view, measuring how much of the functionalities are delivered to the customer. In other words, it calculates the number of functions in a software that meet users’ requirements and are meaningful to the user. FPA also have standardized definition for a smallest unit of function and user requirement. FPA divide functions into data functions and transactional functions. Transactional functions include three types: external inputs, external outputs, external inquiries, while data functions consist of two types: internal logical files and external interface files. In addition, an Elementary Process is a smallest unit of functional user requirement that constitutes a complete transaction and is self-contained.

The concept of case-based reasoning is to solve the current problem using the solution of similar problems that have been solved in the past. Case-base reasoning have a variety of applications in computer reasoning, and could be used in software development measurement and assessment. Case-base reasoning has four steps: retrieve, reuse, revise and retain. First, retrieve the records of similar previous problems after the target problem is given. Then, perform mapping from previous cases to the target case, some adaptations may be involved. After that, the generated new solution is revised by testing or simulation. If the simulation do not achieve the desired result, the new solution could be altered and revised again. Finally, the solution is to be presented and stored as a previous case.

Some artificial intelligence approaches are also suitable for analyzing developer behavior as they try to mimic human behavior and simulate human decision-making process.

In recent years, neural network models have been discussed in the used of measuring software development process. Neural network models simulate the human neural network, it can be used to perform predictions based on a large amount of input data. After the data collected by metrics are fed into the network, it will train itself to give each metric different weight or significance and create layers based on the inputs to find the hidden patterns. In a study by Nassif et al (2016), the cascade correlation neural network model performed the best in software development effort estimation among the four models tested.

The idea of fuzzy logic is inspired by how humans deal with vague concepts and make decisions. In other words, human decision-making do not follow a binary rule, for example there is no such definition as anyone whose height is less than 160cm is short and anyone whose height is more than 160cm is not short. Different people have different perception of height, and there are no strict borders between the definition of short, medium and tall. Fuzzy logic simulates such process and assess the levels of short, medium and tall to make a decision. First the input crisp data is fuzzified, the values of the levels of height is generated, which are between 0 and 1. Then the system would inference a fuzzy output based on the rules defined. Finally the system would defuzzify the fuzzy output to produce an accurate output. Fuzzy logic can be useful for dealing with noisy data and solving problems closer to human life, but it may not produce the most precise and accuracy results.

Natural language processing can be significantly beneficial in analyzing developer communications. Natural language processing is to train computers to process and analyze data on natural human language. The Latent Dirichlet Allocation(LDA) developed by Blei, Ng and Jordan (2003) is a model that uses a multi-level hierarchical model using Bayesian Theorem that produces explanations of the occurrence of similar data, which can be used to output the possibilities of the topics mentioned in a file. For example, LDA can be used to analyze developers’ conversations online, generate the level of relevance of the conversation to their current work. In addition, it can also calculate the efficiency of the conversation together with metrics related to commits, for example if many relevant conversations happened when many useful commits are made, then the conversation is considered to be efficient. It can help measure the more organizational related part of a software engineer’s work.

**Ethical concerns**

Although performance measuring software have only been in the market for merely 4 years, it has had a significant impact on the software engineering industry, they are being used in almost every major company and have proven to be very beneficial. According to GitPrime’s case studies, managers think the data analysis and visualization provided by such software gives a new perspective to think about productivity, it encourages high quality work and allow managers to spot problems in an relatively early stage. However, many still debate on whether developer productivity can be measured and if such software or techniques should be introduced into the workplace. When introducing software like GitPrime to their engineers, many companies report that employees feel skeptical about being measured and analyzed in the beginning. The main concerns of engineers include the data transparency, the measuring algorithm and the impact it will have to the workplace environment.

Data transparency is required for such performance analytics software, but in some cases, data transparency can lead to privacy issues. Due to the mass amount of data collected by these software, engineers may feel a violation in their privacy to some extent. Some may think that some of their private information would be seen by other people, while some may argue that they do not understand where do this information about personal accounts go and how they are analyzed. Thus, they may lead to developers’ not agreeing to being measured for their performance.

After the data is collected, the software would perform analysis and generate report suggesting employees’ productivity. Thus, some developers’ feel skeptical towards if machines and algorithms could measure human behavior accurately. The process of the algorithm handle different data, extract useful information and produce problem suggestions is not transparent, and the algorithm itself may misjudge the situation thus the suggestions may be misleading. Questions about the algorithm could lead to skepticism among employees.

Moreover, developers are concerned about how adapting to such method could affect the working environment. Since performance is being measured and every person has some kind of score on their productivity, it may encourage a competitive and unhealthy work environment where people try very hard to become the top engineers in their team, as well as feeling self-conscious and anxious when they are not in the top. If the analyzing algorithm is not objective enough, it may encourage employees to perform activities that are more beneficial to their score, which may not be beneficial to the project being developed. In other words, developers may become overly focused on statistics rather than the project itself.

Several solutions have been brought up by managers in GitPrime’s customer studies to cope with developers’ skepticism that produced desired result.

Being transparent is considered to be one of the most important and effective solution. Employees should be aware of how they are measured from the beginning of adapting to such software. Managers could go through the software with developers’, discovering what activities would be measured and how they impact the overall score. In other words, developers should understand what data are used, and the specific activities that lead to specific analysis output.

Another solution is to make the software adapt to the specific company situations. Managers and employees could discuss what metrics can be omitted and what metrics could have more weight or less weight in the analysis if they can be changed. By altering the algorithm to fit the development process of specific projects, it would become more objective in the scope of the project team, making more accurate analysis on collected data.

Managers should also make clear that a higher measurement could not be achieve with a few metrics, it is the combination of many aspects of the work process and developers should not be focused on statistics. In addition, managers could make employees know that the software could not replace a manager, it only gives suggestions and managers make the final decision. As well as setting rules and regulation to protect employees’ privacy.

In addition, managers should slowly adapt the software. For example, only using several metrics and use one kind of report to make decisions in the beginning, then making use of more functionalities after team members see the result and agree that it is beneficial.

A theory of using Gamification to boost productivity is also suggested by GitPrime. Gamification is to apply game-design elements and gaming principles in non-gaming context according to Wikipedia. In other words, Gamification is to let developers treat working as a game, the reports and measurements as scores in the game, and the goal of the game is to maintain a good mark. By maintaining a high mark, it shows that developers are focused on their work and delivering efficient codes. A 2012 study conducted by Singer and Schneider on computer science students by using social application to motivate students showed that the web application have made students commit more frequently and commit smaller changes. The web application used in the study was relatively simple, it only monitored a single metric of the number of commits per person, created a web newsfeed for team members to communicate, sent out weekly report emails and a milestone system to set goals to students. Students report that the milestone system and newsfeed platform made them want to achieve the next milestone and not fall behind on other teammates. Though the experiment may be small, but the result shows that adapting gamification gave students motivation to contribute more to the coursework and commit more often, which was the researchers’ goal. In conclusion, treating performance measuring software as a game could make developers more motivated to produce quality work.

**Conclusion**

Though many debates on whether there could be an appropriate tool to measure and assess developer productivity, I believe it can be achieved. A large amount of data should be collected and the assessment is the result of many different metrics in different aspects, the idea of a few “magic metric” that could solve the problem do not exist. Many companies are adapting this new method by using platforms such as GitPrime, Velocity, GitClear and Pinpoint. The analysis process can integrate many algorithmic approaches including regression model (COCOMO), function point analysis, case-based reasoning, neural network model, fuzzy logic and natural language processing to achieve an accurate analysis on developers’ work. Many developers have concerns about privacy issues and the accuracy of the software measurement, but they could be solved by deploying methods that have made success in some companies.

**Resources**

* Blei, D. M., Ng, A. Y., & Jordan, M. I. (2003). Latent dirichlet allocation. Journal of machine Learning research, 3(Jan), 993-1022.
* Fenton, N. E., & Pfleeger, S. L. (1998). Software Metrics: A Rigorous and Practical Approach: Brooks.
* Gaffney Jr, J. E. (1981, January). Metrics in software quality assurance. In Proceedings of the ACM'81 conference (pp. 126-130). ACM.
* Gousios, G., Kalliamvakou, E., & Spinellis, D. (2008, May). Measuring developer contribution from software repository data. In Proceedings of the 2008 international working conference on Mining software repositories (pp. 129-132). ACM.
* Hélie, J., Wright, I., & Ziegler, A. (2018). Measuring software development productivity: A machine learning approach. In Proceedings of the Conference on Machine Learning for Programming Workshop, affiliated with FLoC (Vol. 18).
* Johnson, P. M., Kou, H., Agustin, J., Chan, C., Moore, C., Miglani, J., ... & Doane, W. E. (2003, May). Beyond the personal software process: Metrics collection and analysis for the differently disciplined. In 25th International Conference on Software Engineering, 2003. Proceedings. (pp. 641-646). IEEE.
* Nassif, A. B., Azzeh, M., Capretz, L. F., & Ho, D. (2016). Neural network models for software development effort estimation: a comparative study. Neural Computing and Applications, 27(8), 2369-2381.
* Park, H., & Baek, S. (2008). An empirical validation of a neural network model for software effort estimation. Expert Systems with Applications, 35(3), 929-937.
* Sadowski, C., & Zimmermann, T. (Eds.). (2019). Rethinking Productivity in Software Engineering. Apress.
* Singer, L., & Schneider, K. (2012, June). It was a bit of a race: Gamification of version control. In 2012 Second International Workshop on Games and Software Engineering: Realizing User Engagement with Game Engineering Techniques (GAS) (pp. 5-8). IEEE.
* 2017 Software Developer Productivity Survey | GitPrime Blog

<https://blog.gitprime.com/2017-software-developer-productivity-survey/>

* GitPrime Raises $10.5M in Series A

<https://www.vcnewsdaily.com/gitprime/venture-capital-funding/yktgxrfjpc>

* Harness the potential of your engineering team - GitClear

<https://www.gitclear.com/>

* Line Impact factors - GitClear

<https://www.gitclear.com/line_impact_factors>

* Measuring Developer Productivity A Comprehensive Guide For The Data Driven - GitClear

<https://www.gitclear.com/measuring_developer_productivity_a_comprehensive_guide_for_the_data_driven#footnote_1>

* Customers | GitPrime

<https://www.gitprime.com/customers/>

* Velocity vs. GitPrime: Choosing an Engineering Intelligence Tool

<https://codeclimate.com/blog/velocity-vs-gitprime/>

* Engineering Performance Management Software – Pinpoint

<https://pinpoint.com/>

* COCOMO - Wikipedia

<https://en.wikipedia.org/wiki/COCOMO>

* Software Engineering | COCOMO Model - GeeksforGeeks

<https://www.geeksforgeeks.org/software-engineering-cocomo-model/>

* Case-based reasoning - Wikipedia

<https://en.wikipedia.org/wiki/Case-based_reasoning>

* Can’t You Just "Game" the Metrics? - GitPrime Help Center

<https://help.gitprime.com/general/cant-you-just-game-the-metrics>

* Gamification - Wikipedia

<https://en.wikipedia.org/wiki/Gamification>

* Fuzzy Logic | Introduction - GeeksforGeeks

<https://www.geeksforgeeks.org/fuzzy-logic-introduction/>