Measuring Software Engineering Report

Name: Shengyuan Liu

Student Number: 16341982

Lecturer: Pro Stephen Barrett

Date: 22/10/2019

Specification

Deliver a report that considers the ways in which the software engineering process can be measured and assessed in terms of measurable data, an overview of the computational platforms available to perform this work, the algorithmic approaches available, and the ethics concerns surrounding this kind of analytics.

Introduction

Nowadays, with the rapid development of the Internet, more and more software has been created and used in people's daily life. The need for new software is growing faster and the various functions that different software can realize make people's information life become more and more colorful and convenient. Software engineering has also become a necessary subject in universities, especially those focusing on science and engineering courses. It can be said that human life is inseparable from software engineering.

However, according to the Stack Overflow Developer Survey Result 2019, there are more than 51.5% developers code less than 10 years and 41% of the respondent code professionally less than 5 years. Therefore, whether for software development team or an individual software engineer, it is very important to have a complete, objective and effective evaluation system or approach to evaluate the software development process. These approaches should not only tell Software development company what contribution each developer is making in software development, whether some developers are lazy or not, but also give positive feedback to the team or workers. Software managers can use software metrics to identify, prioritize, track, communicate any issues to foster better team productivity. This enables effective management and allows assessment and prioritization of problems within software development projects. The sooner managers can detect software problems and solve them in the early time. Software development teams can use software metrics to communicate the status of software development projects, pinpoint and address issues, and monitor, improve on, and better manage their workflow. Although it is easy to quantify the work of the other subjects' field such as finance and insurance which evaluation metrics are sales amount and pure profit etc. In Software engineering there are a plenty of different metrics to evaluate the software developers' work. But none of them can provide a complete and effective measurement overall. This report will introduce some methods and evaluate them in different ways.

Ways of Measuring Software Engineering

Measurable data/Simply method

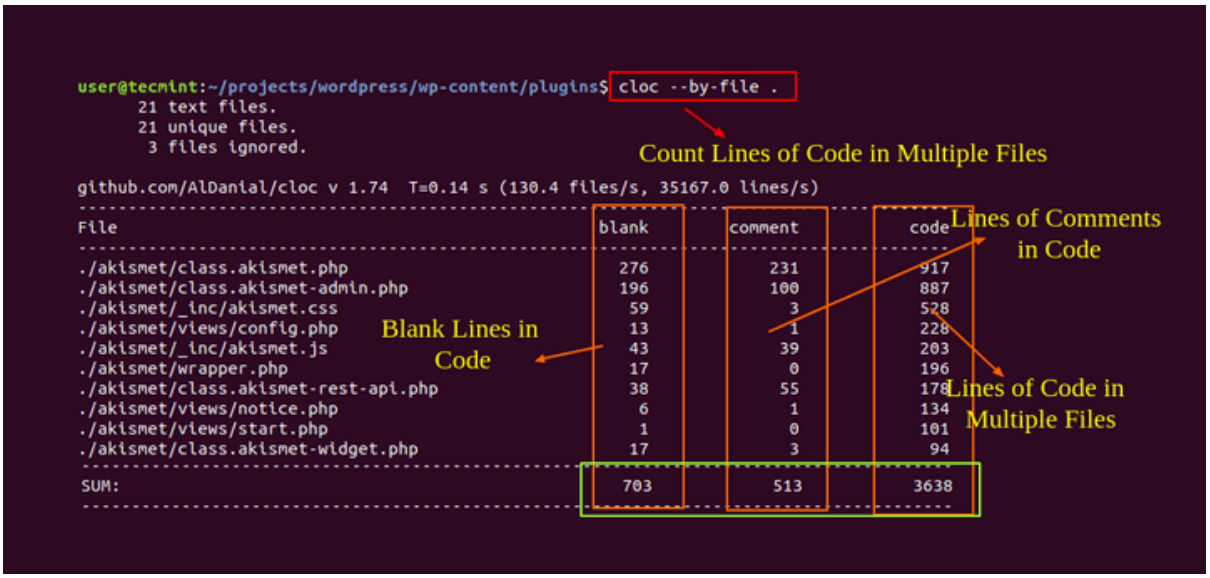
As the introduction said, there is not a metrics for the Software Engineering development to measure the develops' work completely, objectively and effectively. So, I start from the basic and simple evaluate methods which can use the measurable data for each software engineering developer.

The most used and popular of these measuring methods is called Source Lines of Code (SLOC). This method was created in the early time which the program and code just be created. It is used to judge each software engineer's contribution to the whole development by count each physical line that ends with a return which he/she submitted into the project. SLOC is also typically used to predict the amount of effort that will be required to develop a program, as well as to estimate programming productivity or maintainability once the software is produced.

As you can see, this method is so basic and simple which can only be used at the beginning of the birth of computers and programming. Because At that time, there weren't so many different programming languages in the programming world, every developer used the same program language. Therefore, judging the amount of code was the most intuitive way to judge a software engineer's workload. Its advantage is simply and straight forward, the physical code numbers is a real metric for every programmer. When the program language is the same, who write more, who get more is logical and everyone admit it.

However, the disadvantages of this method are too obvious, Nowadays, with the rapid development of programming technology, a variety of programming languages have been developed and used by people, and each language corresponds to different application fields and software applications. Just implement the same function in the different language, the low-level programming languages have twice as much or more code number than the high-level ones, the developers who use the high-level programming languages may use more time to think rather than write. Therefore, it is unsuitable to measure the contribution of a software engineer only by the amount of his/her code. There is also an issue with how software metrics are used. If a software organization uses productivity metrics that emphasize volume of code and errors, software developers could avoid tackling tricky problems to keep their SLOC up and error counts down. Software developers who write a large amount of simple code may have great productivity numbers but not great software development skills. This is a great harm to the actual efficiency of software development in the real life.

The platform to obtain the SLOC is called the CLOC (Count Lines of Code). The CLOC counts blank lines, comment lines, and physical lines of source code in most of the major programming languages those developer use. It is written entirely in Perl with no dependencies outside the standard distribution of Perl v5.6 and higher and so is quite portable.

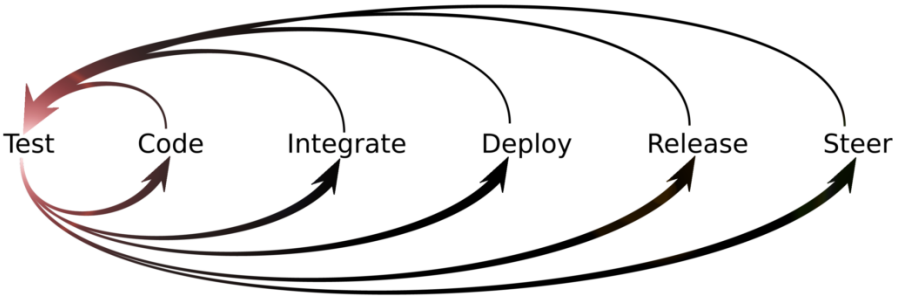


A basic interface of the CLOC

With the development of The Times, the amount of code is less and less representative of the effective workload. So, the SLOC is increasingly being used to estimate the size of entire software so that teams can assign and schedule tasks to the developers.

As time goes on, the software development team found it unreasonable to judge a developer's contribution to the overall software by the amount of code, so they used a very simple method which measure the test number of the software by each developer. Then the TDD (Test-driven development) had been created. More and more software engineering development companies or teams are using TDD or ATDD for software development because it has many advantages for developer. First, it can reduce a lot of the rework time because the developer TDD advocates to write the test program first and then code to implement its function. Therefore, it is rare to write code and find that the test objective cannot be achieved. The manager is also easy to count every developers' workload and their contribution which just need to calculate their test number, the test time and the difficulty and complexity of the test (this would Have certain subjective tendency). Second, it can effectively provide software engineering

developers from the waste of time caused by excessive design, many software developers always plan a big blueprint before writing the code and build a plenty of useless function cause delay in processing. And TDD allows developers to have clear goals of their work and confirm own workload at any time. At last, TDD gives developers a more comprehensive view of development, group manager also can arrange the develop time with ease. That is why more and more software engineering company and developer team use it to Improve their work efficiency. And to measure the time which each developer spend on the TDD work is a better and more suitable method to judge their contribution to the whole software develop.



The basic process steps of TDD

To implement the measurement of the TDD uses 5 different indicators which are 1. Time of running and manually testing your application: measure the time it takes to run the same software and test it manually. With a developer who is good at using TDD it will decrease sharply. 2. Object-oriented design quality: A developer who use TDD Skillfully, his/her work's Cohesion will increase, Method's length and Cyclomatic complexity will decrease. 3. Code coverage: measure Code Coverage for developers' code. A developer who contributes more to the software should get higher percentage of coverage than others. 4. Bugfixing time: measure the time software team reports on identifying, managing and fixing defects. When the developer's productivity is higher, the less time he/she spends on Bugfixing. 5. Average Lead Time: measure the Average Lead Time for each developer's project. Expect it to go down with who is good at using TDD.

The software which obtain the TDD are xUnit frameworks and TAP (Test Anything Protocol) results. Although the TDD has so many advantages and a complete system with standard test indicators of the developers and software. It still has its own disadvantages: First, the developer may only complete the code that meets the test and neglect to implement the actual requirements. Second, it could slow down the development of actual code, especially for prototypes that require speed of development software. Third, For GUI, databases, and Web applications software. Unit tests can be difficult to construct, and if developers force the tests to be constructed, they put extra effort into maintenance. Fourth, TDD can lead to a lack of coverage of unit tests, such as a possible lack of boundary testing.

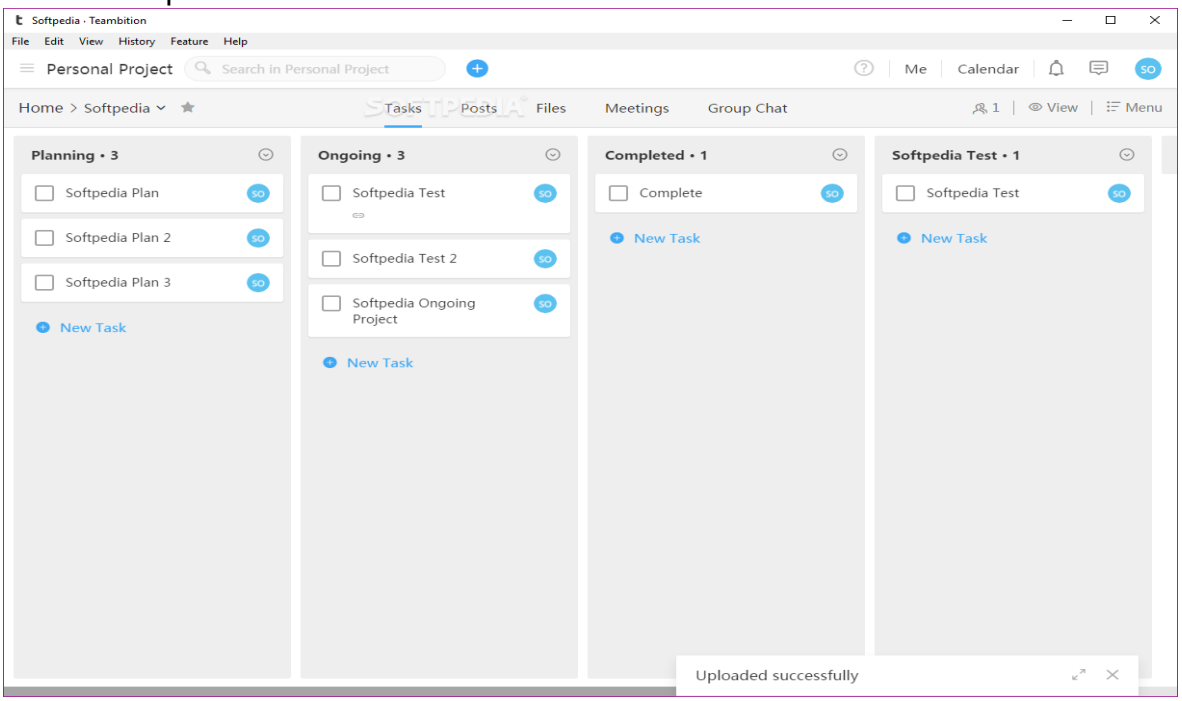
Hence, as above said, until now, there is still no any methods can measure a software development process, the productivity and effective efficiency of developer's work completely, objectively and effectively on its own.

Computational Platforms

According to the results of the previous section, neither simple methods nor measurable data can evaluate the contribution of a software engineer to the whole software development and his/her developer's working efficiency perfectly. Therefore, engineer created a variety of basic and advanced platforms to measure as well as possible the progress and productivity of a software development team or individual workload.

Personally, I was very lucky to have the opportunity to work as an intern in a Chinese software development team which developed and operating mobile applications in the summer vacation of my sophomore year. And my job is software quality control intern. So, I have some very basic knowledge and experience of software development measurement, the measurement computational platforms which my company used was the TB (Teambition). As the figure shows below, the platform will first allow all software developers to register, and then the team leaders will carry out the initial blueprint construction. After the goal of software development is established, each software engineer makes time plans for himself, and these plans are marked as small events and marked with time stamp. These small events can be viewed on the platform in the form of tags and each developer's event tag can be moved across and stay under the different large progress sections as the individual workload. When a developer's event has been completed or moved, the platform records how much time the developer has spent. And the manager of the software team or company can use these data to calculate the actual work efficiency or KPI of each software engineer as well as measure the whole team's workload is in line with expectations or not. The data on the platform are more intuitional to view on the form or table. One of the great advantages of this platform is that it is very time sensitive. Any activity that is already on the schedule or new temporary events such as rework the code which does not conform with the demand can be added to the platform at any time by any developer. And in my opinion, this platform to give me the most convenient and most important function is the platform incorporates a lot of different simple measuring method and measuring data, so every time after completion of a large software development phase, the platform will automatically according to he can the collected data and own evaluation method for platform system, I give every software engineer including automatic generate a test

report let me know which several stages in the development process has delayed my actual work efficiency or does not meet the normal index. With such reports, each software engineering developer can have a good understanding of their own development speed and work efficiency and can have a clear target to improve their own develop weaknesses.



The Teambition platform main interface

However, although we have such a computational platform for software engineer, the team and the company still cannot say they can completely, objectively and effectively measure software engineering process. The mainly reasons are following: 1. Lack of the measurable data or methods, thought the platform collect a lot of data and use many different methods to measure software engineering process, compare with all the measurement methods in the world those data and methods are just a drop in the bucket and so far from enough. 2. Human factors: Although the report is generated automatically by the platform through the methods of measurement data, human factors mainly influence on the measurement of time, for example, sometimes a company has been off work and developers work overtime on realization of some function units of the software and the submit his/her work. but these functions are conformed the requirements or not need to evaluate by other teams, and the other teams have left the company and cannot evaluate immediately. The platform only computes the time after the other team confirm the functions implementation are in line with the indicator. This may lead 10-15 hours statistics delay when work is already done. And the computational efficiency result is greatly lower than the actual efficiency.

Similarly, there are opportunistic developers who can pretend to complete a feature and let the platform record the time before writing the code. So, the human force makes it impossible for the platform to evaluate the speed and efficiency of software engineering development objectively. 3. Privacy issues: No one likes being controlled by the others as well as monitor, use this platform will makes them feel like they're being monitored and feel uncomfortable.

In general, such a computational platform can provide a relatively complete and objective evaluation for a software engineering team or an individual developer then output an intuitive and targeted report. We have made a great progress of our goal of being able to construct a comprehensive evaluation of the software engineer development process. But it is far from successful or even perfect.

The first person in the world to propose the concept of "computational platform" is Watts S. Humphrey who was also called as the "father of software quality". He also created the first computational platform which was called PSP (Personal software process) in 1996 in IEEE Software during 1996. The PSP goal are helping the software engineers to: Promote their estimating and planning skills, make commitments which they can keep, manage the quality of the developer's projects and reduce the number of defects in the development. The PSP has its own structure which are: PSP0, PSP0.1 (Introduces process discipline and measurement), PSP1, PSP1.1 (Introduces estimating and planning), PSP2, PSP2.1 (Introduces quality management and design) and PSP3 (introduces cyclic development). The measurable data and the measure methods are the most important things of PSP. PSP data collection is supported by four main elements: 1. Scripts. 2. Measures. 3. Standards. 4. Forms. And the Core measures of the PSP are : Size: the size measure for a software section, such as lines of code(SLOC). Effort: the time required to complete a part or process, usually recorded in minutes. Quality: the number of defects in the software. Schedule: a measure of software progression, tracked against the team plan and actual completion dates.

For all the current software engineering computational platforms, the most important significance of PSP is that PSP provides the three most important measurements data which are: time, defect and size data. This evaluation standard has become the mainstream evaluation index of all computational platforms and affects all the measurement core data of later generations. Of course, in today's world, the PSP's evaluation methods and data collection are too simple or even primitive, but without the PSP as the foundation of software engineering computational platform, there would be no so many various computational platforms for our using. And developers can use these platforms to measure the software development process convenient should be grateful to those predecessors. Without their contributions to the discipline of software engineering, we would not be learning and using these platforms today.

Algorithmic Approaches

Finish with the measurable data and computational platform I have already covered. As a discipline of science and information technology, software engineering also has many algorithms to measure the development process of a software and to display the individual contributions of each software engineer more rationally with numerical results or formulas. So, I will introduce two of those algorithms which one is the most popular and most widely used by software engineers is the COCOMO (Constructive Cost Model). And another one is the Putnam model which is an empirical software effort estimation algorithmic model and has a close relationship with COCOMO.

For the COCOMO, it was born in the late 1970s and its inventor was Barry W. Boehm. The COCOMO was published in Boehm's 1981 book Software Engineering Economics and be used as an algorithmic approach for measure the estimation effort, cost, and schedule for software projects.

COCOMO can be divided into two categories which one is the basic COCOMO and the another one is the intermediate COCOMO. Basic COCOMO is a static single-value model and it uses program sizes to measure in terms of thousands of source lines (KLOC) to calculate the effort and cost of software engineer development. COCOMO can be applied to three different software projects: First: Organic projects: relatively small, simple software projects which are done by small, experienced teams, with fewer requirements and less stringent constraints. Second: Medium separation projects: software projects of medium size (size and complexity) which are completed by teams of people with different levels of experience, with both strict and less stringent requirements. And the third: Embedded projects: the software projects which must rely on a compact set of hardware, software, and operational constraints.

The basic COCOMO equation is as follows:

$$E=a_b(KLOC)^{b_b}$$
$$D=c_b(E)^{d_b}$$
$$P=E/D$$

Where E refers to the workload calculated by " person month", D refers to the accumulated development time (months), KLOC refers to the estimated number of lines of code finally released (one thousand lines of code) and P refers to the number of people required. The basic COCOMO is good for quick, early rough estimates of software costs. However, it does not consider the different hardware conditions, personnel quality and experience, the use of modern tools and technologies, and other

project attributes that will have a profound impact on the cost of software, so its accuracy is limited. Software engineering developers almost no longer use it as an algorithmic approach to measure software progress, replacing it with intermediate COCOMO.

Intermediate COCOMO estimates of software effort as function of program size and a set of "cost drivers," including objective evaluations of product, hardware, people, and project attributes. Each class has a few small attributes:

1. Product attributes: Software reliability requirements, Size of application database, Product complexity. 2. Hardware properties: Runtime performance constraints, Memory constraint, Virtual machine stability, Response time requirements. 3. Personnel attributes: Analysis ability, Software engineering ability, Application experience, Virtual machine experience, Programming language experience. 4. Project attributes: Software tools adopted, Application of software engineering methods, Development schedule requirements.

Each of these 15 attributes receives a six-point assessment, ranging from "very low" to "very high" (importance or size). The available factor values are listed in the following table. The product of all these factors is the effort adjustment factor (EAF). And the software engineering would use this product to judge each developer's contribution.

The calculation formula of intermediate COCOMO is as follows:

$$E = a_i (KLoC)^{(b_i)} . EAF$$

Where E is the workload calculated by "person months", "KLOC" is the number of lines of code released by the product (1000 lines of code), and "EAF" is the factor calculated by the above method.

Compare to the basic COCOMO, the Intermediate COCOMO has more detailed evaluation criteria and the measured data are far more than basic COCOMO. Hence the overall measurement results are completer and more objective than basic COCOMO and can be applied in a wider range of fields.

After introducing COCOMO, I want to talk about another algorithmic approach which is closely related to COCOMO and its name is Putnam model. The Putnam model is an empirical software effort estimation model which published by Lawrence H. in 1978. It had been used to measure the Related workload, software working curve and Predict future workload by collecting software project data. And SLIM (Software Lifecycle Management) is the name given by Putnam to the proprietary suite of tools which has developed based on his model. It is one of the earliest of these types of

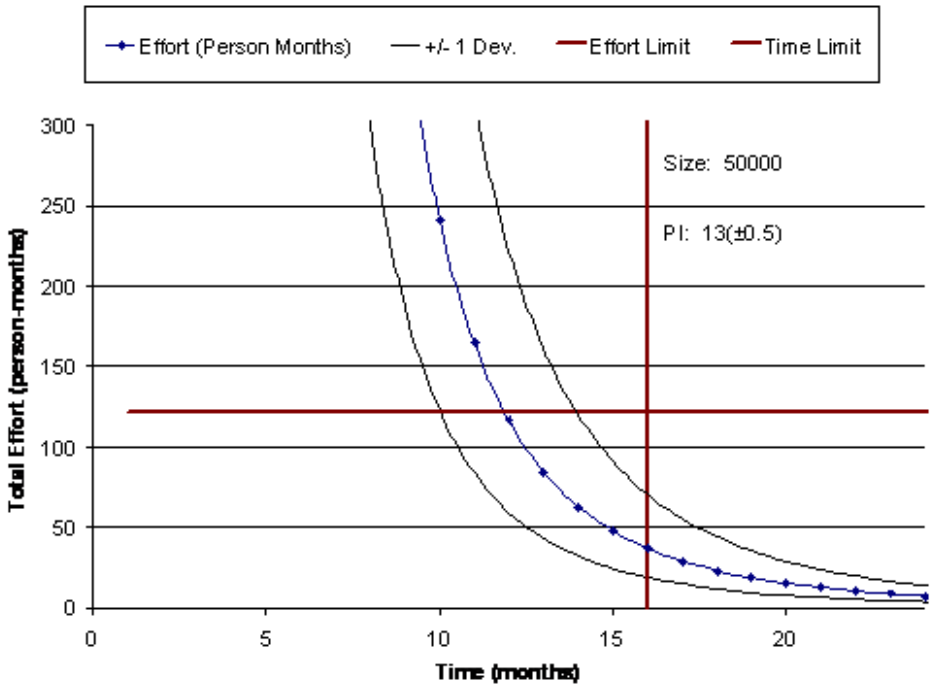
models developed and is among the most widely used. Putnam used his observations about productivity levels to derive the software equation:

$$\frac{B^{1/3} \cdot \text{Size}}{\text{Productivity}} = \text{Effort}^{1/3} \cdot \text{Time}^{4/3}$$

Where the Size is the product size (whatever size estimate is used by your organization is appropriate). B is a scaling factor and is a function of the project size. Productivity is process productivity, the ability of a particular software organization to produce software of a given size at a specific defect rate. Effort is the total effort applied to the project in person-years. Time is the total schedule of the software in years. In actual use, when estimating software tasks, it is necessary to solve the software equation. Hence the equation would change to this form:

$$\text{Effort} = \left[\frac{\text{Size}}{\text{Productivity} \cdot \text{Time}^{4/3}}\right]^3 \cdot B$$

After using the equation of the Putnam model, we could get a time-effort curve graph which is the most important result of this algorithmic approach.



Use software sizes and productivity of organizational processes estimated at project completion. Plotting workload as a function of time yields a time workload

curve. The points on the curve represent the estimated total amount of work to complete the project at a certain time. One of the salient features of Putnam model is that the total effort decreases as the project is completed. And one of the main advantages of the model is simple calibration. Most software, whatever its maturity, can easily collect past project sizes, workloads, and durations. The process productivity is essentially exponential in form. So, it can be typically converted to a linear productivity index that organizations can use to track their productivity changes and detect to the future work trend.

However, the reason that algorithmics approaches have so many advantages but still been not used as a primary test method in real software engineering development today is the algorithmics approaches is a so abstract thing. Therefore, when it is applied to the real software engineering development, it will encounter a lot of unpredictable factors, which will eventually lead to the actual results and the expected results and the expected results which measured by the algorithmics approaches are not the same or even a great deviation. And the algorithmics approaches still cannot measure a software development process, the productivity and effective efficiency of developer's work completely, objectively and effectively for the software engineering team or company.

Ethical Concerns in Measuring Software Engineering

Although I have introduced so many different types of ways or methods to measure the software engineering process, there is a most important problem that is the ethical concerns of these methods or measurements. Whatever for the manager of the software engineering team or the individual developer. In my opinion, a fine balance should be struck between the developer's inherent rights and their privacy and the benefits of software development of the employer. The manager or the employer could collect the data which only developed in the course of the company's work to measure the developer's working efficient and his/her contribution of the but strictly prohibited to collect the data beyond software development. Especially the personal life of developers after his/her own work. Most software engineers follow a code of ethics, which the most notably is the IEE/AMC joint Code of Ethics 1999. And this should be the basic code of ethics for whatever the software developer or the employer of the workers.

No one likes to be monitored, but the truth is that we all live in a surveillance world every minute of every day. From the web pages you visit to your bank account number and password, we all live in a surveillance world. And these monitored also can be divided into good and bad.

In terms of my internship experience, my software development company will make us to register and use several different kinds of office software and collect the working data from these software which we are working or our activity such as our working time, number of each developer rework times, etc. and summarized them Then the company manager according to these data creates an individual report which can tell us which parts are our weaknesses or shortcomings per week. I personally don't dislike this kind of monitoring, but think it is a sign of the company's responsibility to developers. Because these data are only used in the work, which will not affect the privacy and security of the developers themselves, and the employers also pay employees. Therefore, it is reasonable and legal to use these data to measure the progress of software development and the effective working efficiency of the developers.

But equally, there can be some objectionable monitoring data exist, for example, one of my internship using office software will require me to punch in when I go to work or leave, and the clock in's standard is to open my mobile phone positioning system, GPS will automatically detect the individual's mobile phone location is within the scope of the company's coordinates, it will not punch in if the location is out of the scope. And this measurement would make me think there's a risk of losing personal location privacy, so I only allowed the GPS working after arriving the company.

Therefore, the software company should always observe the code of ethics when monitoring and collecting the work data of employees. The data generated by employees at work can be measured or counted by any method or algorithmics approaches, but once the developer is off work, any other data generated by him/her such as the website cookies, personal locations etc. is not allowed to be collected or counted. Only in this way, we can ensure that the privacy of developers has not been violated, the security of personal information has been protected, and developers can work more reassuring for the company.

Conclusion

Overall, Software Engineering is so different from other professions in that you are creating systems to solve problems which are usually not clearly defined. Although I have talked so many measurable data, computational platforms and algorithmic approaches. There is no theory or platform has dared to claim that it can develop a complete, objective and effective measurement software engineering Until now. In my opinion, it is unrealistic to measure software engineering development in one or more ways. The best approach is to use multiple platforms or theories, measure the progress of development, the developer's efficiency and contribution systematic to get the complete and objective measurement as much as possible. But whatever the measurement is, it should obey the code of ethical.

Reference List

M Squared Technologies LLC (2006, November 13). Effective Lines of Code eLOC Metrics for popular Open Source Software, from https://web.archive.org/web/20061113145117/http://msquaredtechnologies.com/m2rsm/rsm_software_project_metrics.htm

Matthew, J. Jan, S. Yves, Z. (2015, October 19). Network Effects on Worker Productivity from https://www.researchgate.net/profile/Jan_Sauermann2/publication/285356496_Network_Effects_on_Worker_Productivity/links/565d91c508ae4988a7bc7397.pdf

Danial, A. (2006). Count Lines of Code, from http://cloc.sourceforge.net/

Beck, K. (2002, November 8). *Test-Driven Development by Example,* ISBN 978-0-321-14653-3, from https://openlibrary.org/books/OL9766821M/Test_Driven_Development

Teambition. (2017). Teambition introduction, from https://us.teambition.com/tour?region=us

Watts S. H. (2005, March). PSP: A Self-Improvement Process for Software Engineers, ISBN 0321305493 from https://resources.sei.cmu.edu/library/asset-view.cfm?assetID=30595

Boehm, B. (1981). Software Engineering Economics, published by Prentice-Hall, ISBN 0-13-822122-7.

Putnam, Lawrence H. (October 1991). Measures for excellence: reliable software On time, within budget, published by Yourdon Press, ISBN 0-13-567694-0, from https://archive.org/details/measuresforexcel0000putn

IEEE. (1999). Code of Ethics, from https://www.computer.org/education/code-of-ethics