Measuring Software Engineering Process

CS3012 Software Engineering Assignment 4

Can ZHOU

19324118

zhouc@tcd.ie

CONTENTS

1 Introduction

Up to now, although there are numerous attempts at measuring the process of software engineering, this issue does not have an official solve approach. In any work driven environment, productivity can be roughly calculated by the equation below:

$$Productivity = Output \div Input$$

Equation 1. Productivity Formula (Spacey, 2017)

But, how to define the input and output? In other words, which metrics should be used? As engineering is complex and multifaced, software engineers' productivity should not only be measured by simple traditional metrics like LOC (Line of Code). This report addresses this topic, focusing on individual productivity and aims to deliver the insight about how to measure the software engineering process from four aspects: 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. In order to understand this topic better, here are some background information on below.

2 Background

2.1 Why measure software engineering process?

The primary purpose of measurement is to provide insight into software systems and products in order to help and allow both supervisor and developer to make decisions and facilitate the accomplishment of objectives. Different identity has different purpose. As the view of project manager, this can help to identify prioritize and areas of improvement, manage software engineers' workloads, reduce cost and overtime, increase return on investment. Most important, it can let manager ensure the team is working on the expecting goal. For Software development teams, they can use software metrics to communicate software development tasks progress, detect and address issues, and track, improve, and maintain their workflow better. (McAndrews, 1993)

2.2 Why software engineering process is hard to measure?

This report is actually based on these questions: why software engineering process is hard to measure? Why productivity cannot be measured by single metric? In my view, software engineering's complexity and multifaced causes main of the issues. The article "*No Silver Bullet – Essence and Accident*" which was written by Brooks have mentioned that there are two types of complexity: accidental and essential complexity. Although accidental complexity can be fixed by engineers, the essential complexity is by the problem to be solved and it is impossible to remove it. (Brooks, 1987) The other one is multifaced feature in software engineering process. This point is that there are just way too many variables and sides and every aspect of software development is fluid and changing. There is not one metric or even a set of metrics engineers can pick out that

will accurately tell engineers anything useful about a software development project. (Sonmez, 2013) I agree with Brooks and Sommez's view and I think engineers cannot make the software simpler (or increasing the productivity) with simpler measurement methods. That is, measuring the productivity by one single metric is obviously incorrect.

3 Measurable Data

There are lots of different types of data being produced when develop software. However, to measure software engineering process needs to determine appropriate measurable data. Measurable data should not be chosen simply because they are easy to obtain and display – only data that add value to the project and process should be tracked. Moreover, there has an argument that traditional metrics is too hard to measure software engineering and should use other fungible measure metrics like happiness instead. In this part, I will contain three kinds of metrics: first level metrics (core metrics) which are countable, second level metrics which are related with individual's changing by time and third level metrics – the patterns which could be used to predicted. I will explain and give the example metrics for each level.

First Level Metrics	Line of Code (LOC)	Number of Commits	Number of Bugs
	Work in Progress (WIP)	Test Coverage Ratio	Function Point (FP)
Second Level Metrics	Code Churn	Pull Request – Related Velocity	Speed of Developer
Third Level Metrics	Happiness - 1/T Rule	Network Size	

3.1 First Level Metrics – Countable

This kind of metrics is the basic metrics which are countable and represented by numbers. Other level metrics are based on this level metrics. As there are numerous metrics in this level, I only can show a small part of them. In the metrics I showed, there have metrics more related with software engineer's personal skills (from 3.1.1 to 3.1.5) and the metrics which are used to detect whether productions meet the requirements (3.1.6).

3.1.1 Line of Code (LOC)

LOC is one of the first metrics used to figure out both software engineer productivity and software quality by counting the number of lines in the code. There are two types: Physical LOC – number of lines of code and Logic LOC – number of executable statements. LOC is a simple metric that widely be used in software engineering, but many computer scholars have questioned this metric, for instance: Bill Gates said: *"Measuring software productivity by lines of code is like measuring progress on an airplane by how much it weighs"* Obviously, this simply traditional metric currently may be not suitable for measuring software engineering. In my view, more code does not equal to higher productivity, as large number of LOC may have many unnecessary codes and long-winded code structure.

3.1.2 Number of Commits

The commits could show the indication of activity. The number of commits is easily tested; however, it could be abused. For instance, if the software engineers have a commit when only a single line of code changed, it will produce lots of non-meaningful commits.

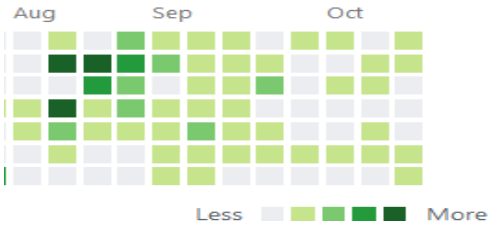

Figure 1. The number of Commits in GitHub

3.1.3 Number of Bugs

The number of bugs shows the system quality which could be counted by the whole system, per line of code and so on. However, this metric may be useless in many situations. An article written by Stackify suggests that software engineers should not spend more than 20% of engineering time, else the program might have a quality/architecture problem and it will be a drain on the productivity. (Stackify, 2017)

3.1.4 Work in Progress (WIP)

Work in Progress is the cumulative number of tickets opened by the software engineer and currently working on them. This metric is useful to understand the current workload of an engineer as a trend and it helps avoid burnout and increase efficiency as the focus has been shown to be improved by working on one thing at a time.

3.1.5 Test Coverage Ratio

Test coverage (also referred to by some as code coverage) is one of many metrics are commonly used in measuring software engineering. It shows the percentage of code which is covered by automated tests. The equation below can be used to calculate the test coverage:

$$Test\ Coverage\ Ratio = \frac{Number\ of\ lines\ of\ code\ called\ by\ test\ suite}{Total\ number\ of\ lines} \times 100\%$$

Equation 2. Calculate Test Coverage Ratio

However, there has an argued that whether the codes which have 100% of test coverage is equal to have high productivity or quality? Cooney pointed out that high test coverage only means maybe the code quality is high, but it does not mean all required functions have been implemented. (Cooney, 2019)

3.1.6 Function Point (FP)

A function point is a unit of measurement to express the amount of business functionality an information system (as a product) provides to a user. In this metric, the functional user requirements of the software are identified and each one is categorized into one of five types: inputs, outputs, inquiries, internal files, and external interfaces. However, function points are not easy to master and varying methods. This is why many managers and teams of software development altogether skip function points. They do not view function points as worth the time.

3.2 Second Level Metrics – Changing by Time

This level's metrics are analysable data which comparing the changes of same metrics in the first level. By comparing changes in metrics at different times, it is possible to detect programmer performance and productivity trends over time. In this way, we can easy figure out whether this engineer's performance is improving, and whether the engineer's continued productivity is stable in a good state as well. Actually, let all the metrics in the first level relate with time could produce data of the changes which can be regarded as second level metrics. I will show three example metrics in this part:

3.2.1 Code Churn

Code churn is used to measure the percentage of a developer's own codes which have modified, added and deleted over a short period. Code Churn is really useful only when its values have an unexpectedly rise or below the individual's 'normal'.

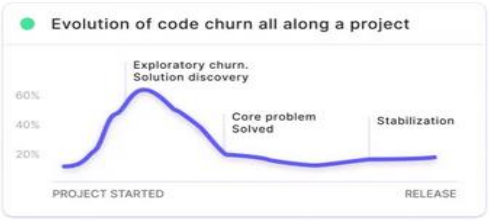

Figure 2. Code Churn

3.2.2 Pull Request – Related Velocity

This metric can be counted as the number of pull requests opened per week, the number of pull requests merged per week and the average time to merge. This metric could give project manager a sense of engineering team's constant productivity.

3.2.3 Speed of Developer

This metric is to figure out how fast the developer can finish the work. If a software engineer can finish more tasks than others in a period or can complete the same task faster than others, it means this engineer's speed is high. It is a good metric to show the software engineer's performance and can compare different developers' productivities in a period of time. However, it should be noticed that, although some engineers can accomplish the work very fast, it does not mean their code quality is high.

3.3 Third Level Metrics – Patterns

The third level is the further development of the first two levels. This metric can be associated with the patterns. Collect data through a wide range and use those data to discover patterns. Like recommendation system, users with the same search data are more likely to have same preferences. So, if the software engineer has similar data for a metric, the performance and productivity of the software engineer can be predicted from pattern. I will show two this kind of metrics: Happiness and Network Size.

3.3.1 Happiness - 1/T Rule

It has been stated by researches that the happiness of an individual affect's performance significantly. Researchers have been found that people who are happy have 37% higher productivity at work and 300% higher creativity compared to people who are unhappy. So, happiness can be regarded as a metric to measure software engineers' productivity.

If a software engineer has a high happiness, he is more likely to have a high productivity. Recently, the researchers are able to relate happiness to physical activity by using the wearable technology. Research has found a latent message reflecting the happiness of an individual within the basic physical activity cycle known as the ' 1/T law. ' This resulted in a close relationship between the 'trinity' of physical activity = satisfaction = efficiency, hence, the data of happiness becomes measurable. (Yano, et al., 2015)

3.3.2 Network Size

Network size is the number of a work's co-worker networks. Lindquist has studied how co-worker productivity affects worker productivity via network effects. They have proved the worker's productivity is related with network size by experiences and it shows that a 10% increase in the current productivity of a worker's co-worker network leads to a 1.7% increase in own current productivity. Moreover, they have determined the pattern between network size and predicted productivity which is represented in the figure. So, the network size is measurable and can be deemed as a metric to measure the productivity. (Lindquist, et al., 2015)

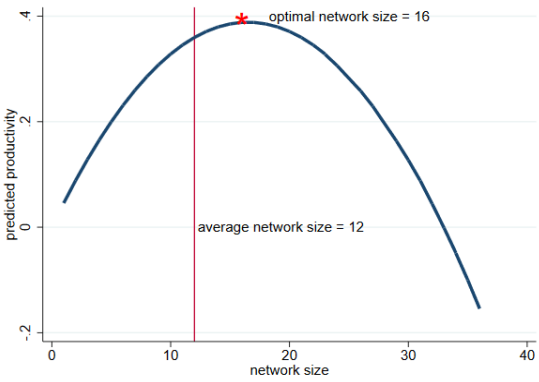

Figure 3. The Relationship Between Productivity and Network Size

3.4 Conclusion for Measurable Data

In this part, I have mentioned three different level of metrics. The first level is core metric which is more refer to countable feature. The second level of metrics is focused on changing by time. Those metrics are related with time to gain the core metrics' changes in a period. So, engineers' productivities are easier to compare and analyse. Then, I explain what is the third level metrics (patterns). We can measure some alternative metrics like happiness to predict productivity from pattern. In a pattern, if the measured data show engineer may have a low productivity, employers can make the act before the situation becomes worse. But we need to note that, like recommended systems, recommendation/prediction is not always accurate, so it may cannot exactly represent engineers' productivity. Patterns still have a lot of room for improvement. Last, we can see each metrics can only concern one aspect of engineering, so using one single metrics to measure the engineer's productivity will not be complete and correct.

4 Computational Platforms

There are a number of computational Platforms available to collect data and measure software engineering process. Here are some platforms which could roughly divide by two categories – free platforms and premium platforms:

Free Platforms	GitHub API	Eclipse Plug-in	Personal Software Process
	The LEAP Toolkit	Hackystat	PROM
Premium Platforms	GitPrime	Code Climate Velocity	
	Codacy	Hubstaff	

4.1 Free Platforms

4.1.1 GitHub / GitLab / Bitbucket API

These version control web-based tools offer all the features and functionalities of Git with many extra services. In the report, GitHub will be chosen as an example. GitHub offers a publicly available API that can be used to request a whole host of data about publicly available repositories.

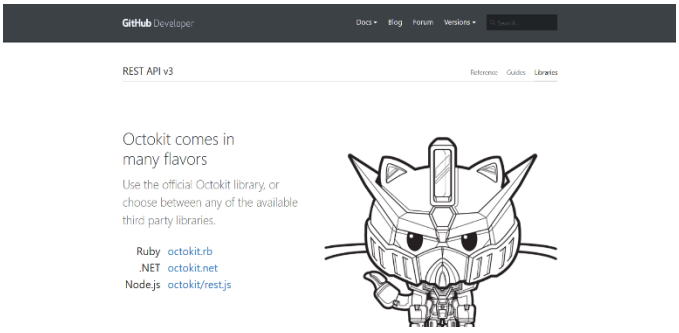

Figure 4. GitHub API

4.1.2 Eclipse Plug-in

The plug-in in Eclipse could help to collect and analysis data as well. One of instance is Mylar Monitor which is a standalone framework for capturing and recording trace data on the operation of a user in Eclipse. The Mylar Monitor records events such as changes in priorities, shifts in orientation, window events, options, inactivity times, commands invoked by menus or button links, and URLs displayed through the Eclipsebrowser installed. (Murphy, et al., 2006)

4.1.3 Personal Software Process (PSP)

The Personal Software Process (PSP) is a structured software development process designed to improve software engineers' the understanding and performance by bringing discipline to the way they develop software and tracking their predicted and actual code development. It clearly shows developers how to manage their products ` quality, how to make a sound plan and commits. It also provides them with the statistics to support their intentions. By analysing and reviewing data on development time, defects, and size, they can evaluate their work and suggest direction for improvement. (Johnson, et al., 2003)

Name: Jill Fonson **Program:** Analyze.java

Date	No.	Type	Inject	Remove	Fix time	Fix defect no.	Description
9/2	1	50	Code	Com	1	1	Forgot import
9/3	2	20	Code	Com	1	2	Forgot ;
9/3	3	80	Code	Com	1	3	Void in constructor

Figure 5. A sample defect-recording log.

Although PSP has above benefits, it also has not negligible disadvantages. Its manual nature resulted in data quality concerns which in turn lead to incorrect conclusions being drawn. As shown in the figure, In PSP, even compiler (syntax) errors are recorded. Developers typically find this aspect of the PSP to be onerous.

4.1.4 The LEAP Toolkit

LEAP addresses the data quality problems that PSP faced by automating the data analysis part. However, this still has to collect the most of required data manually. LEAP toolkit provides a portable repository of personal process data that can be taken by software engineers as they switch from one project to others. This toolkit does not focus on the developers' name but keeps track the developers' activities, the data they added etc. (Moore, 2000)

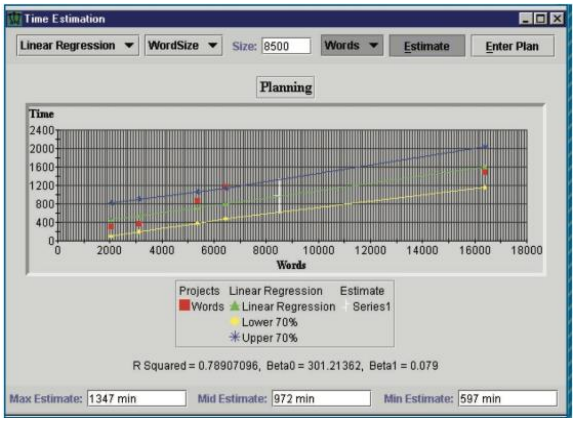

Figure 6. The time estimation component in the Leap. Unlike the PSP, no Leap analytics are paper based.

4.1.5 Hackystat

Hackystat is an open source platform for collection, evaluation, visualization, description, annotation, and distribution of software development system and product data. Users of Hackystat usually add ' sensors ' software to their development tools which unobtrusively capture and send "raw" development information to a web service called the database Hackystat SensorBase. Other web services that query the SensorBase database to create higher-level abstractions of this raw data and/or incorporate it with other internet-based communication or coordination mechanisms and/or produce raw data, abstractions, or annotations visualizations. (Johnson, 2013)

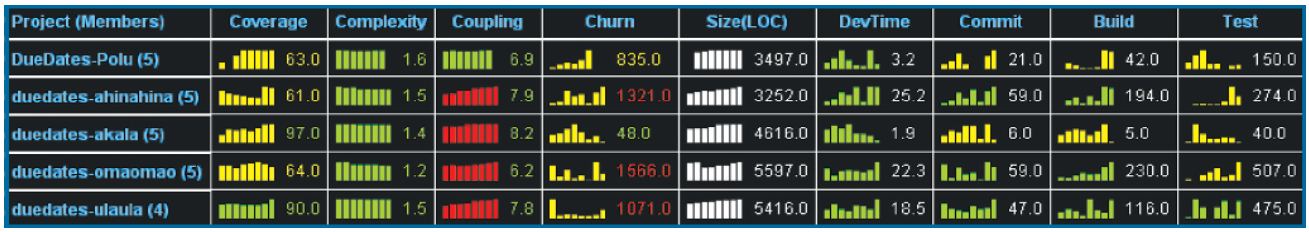

Figure 7. A software ICU (intensive care unit) display based on Hackystat. The Software ICU assesses a project's health both alone and in relation to other projects.

4.1.6 PROM

PROM is an automatic data collection and analysis tool that takes measurements of software and system. The platform relies on comprehensive data collection and evaluation in order to provide criteria of product improvement. The data collected includes a wide range of metrics including all PSP metrics, procedural and object-oriented metrics, ad-hoc metrics developed to trace activities rather than coding with a word processor such as writing documents. The software gathers and analyses information at various granularity levels: personal workgroup, and company. Such distinction takes a picture of the entire software enterprise and protects the confidentiality of programmers that only aggregate data is available to managers. (Sillitti, et al., 2003)

4.2 Premium Platforms

4.2.1 GitPrime

GitPrime is a tool that can draw insights out of git repos and measure software team's success. In order to achieve that, GitPrime aggregates historical git data into easy to understand observations and documents. However, GitPrime require at least USD 749 per month.

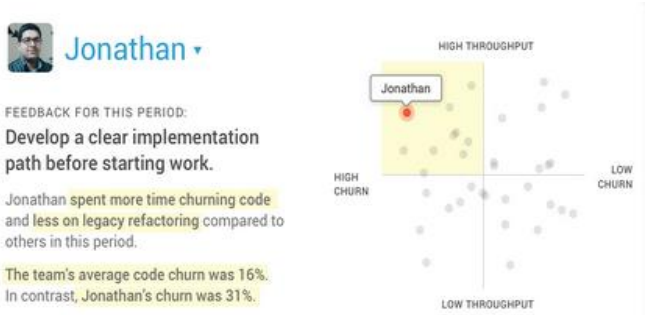

Figure 8. Example of Using GitPrime

4.2.2 Code Climate Velocity

Velocity is out of betaCode Climate includes automated software code review, helping software engineers to fix safety and quality issues before they hit development. It will check every commit, branch and pull request for changes in quality and potential vulnerabilities. Code Climate Velocity will cost at least USD 1759 per month.

4.2.3 Codacy

Codacy can automatically identify issues through static code review analysis. Get notified on security issues, code coverage, code duplication, and code complexity in every commit and pull request, directly from current workflow. It can be taken by free for starting up, but at least USD 15 per user/month when growing team.

4.2.4 Hubstaff

In addition to features in the platforms above, Hubstaff can also tracking time, location, generate report, time scheduling and so on. For one user, basic features can be used for free, but when it comes to more users and features, Hubstaff will charged from USD 7 per user/month.

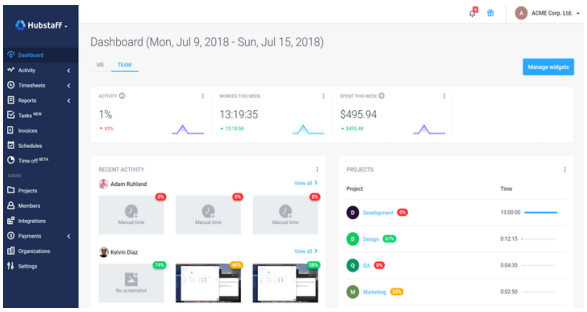

Figure 9. Example of Using Hubstaff

4.3 Conclusion for Computational Platforms

In this part, I have mentioned some useful computational platforms to collect and analyse the productivity. I have divided them into two group by whether the platform is free to use. Each platform has itself specialties and disadvantages and we should be noticed that all platforms have used multiple metrics while they are measuring the software engineering process. It is also proved productivity cannot be measured by single metrics. Moreover, so far, the mainstream platforms are measuring the first and second level metrics, and there are not many platforms to collect and analyse the third level metrics for the patterns.

5 Algorithmic Approaches

There are also have some algorithmic approaches available. I will focus on two algorithms: Bayesian Belief Nets (BBNs) and computational Intelligence (CI) with its three main components: Neural Networks, Fuzzy Systems, and Evolutionary Computation.

5.1 Bayesian Belief Nets (BBNs)

Bayesian Belief Nets (BBNs) is graphical networks linked to a set of tables of probability used to predict software defects. The figure on the right shows the structure of BBNs. Each node represents an uncertain variable and the arcs represent the variable relationships. the probability tables provide the probabilities of each parameter condition for each node. The graph contains a mix of variables that software engineers want to predict. With the proof from Fenton and Neil, the BBN could provide the probabilities of certain outcomes. For example, all probabilities are updated when software engineers enter evidence about the number of defects found in testing. This will always measure that state's probability independent of the amount of evidence given. With greater uncertainty of values, the lack of evidence is reflected. Success in these methods depends on the development process's stability and maturity. The organisation must collect the basic metrics data and perform systematic tests in order to be able to effectively apply this model. (Fenton and Neil, 1999)

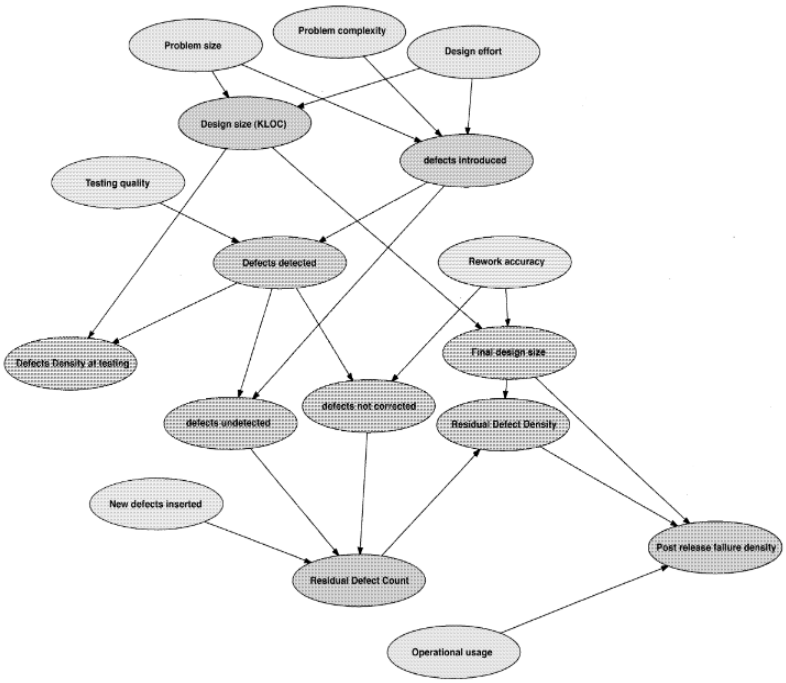

Figure 10. BBNs

5.2 Computational Intelligence (CI)

Computational Intelligence (CI) is a machine's ability to learn from a set of data and/or experimental observations a particular task. The software engineering process is of a random nature and there is an amount of uncertainty during the process. CI is a revolutionary way of solving contemporary, complex issues where conventional computational methods are no longer sufficient. CI has an ability to deal with inaccurate and incomplete knowledge or situations because it attempts to approximate human thought. There are three main components to CI:

Neural Networks (NNs)	Fuzzy Systems (FS)	Evolutionary Computation (EC)

5.2.1 Neural Networks (NNs)

Using the human brain as a source of inspiration, artificial neural networks (NNs) are massively parallel distributed networks capable of learning from instances and generalising. Neural networks have three tandem-functioning components. One processes the information, one sends the signal, one controls the sending signals. It is a distributed system of information that learns from experimental data. Like humans, it accounts for fault tolerance. Neural networks were used in software reliability models development, self-organizing maps in software reusability and software quality models. (Pedrycz and Peters, 1997)

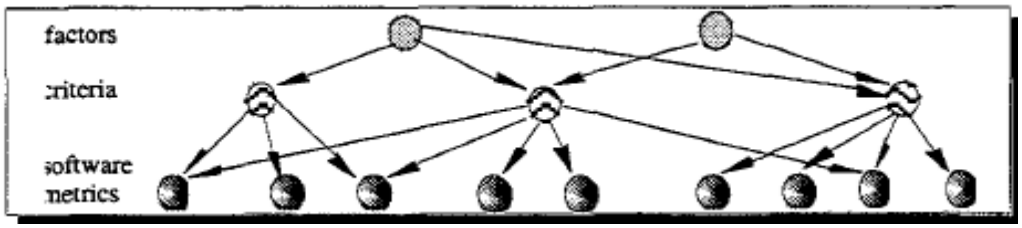

Figure 11. Software Quality Model

5.2.2 Fuzzy Systems (FS)

Using human language as a source of inspiration, fuzzy systems (FS) model linguistic imprecision and solve uncertain issues based on a generalisation of traditional logic that allows to conduct approximate reasoning. In an attempt to make system or process complexity manageable, Fuzzy sets exploit imprecision. The common use of linguistic terms in software engineering can be seen as evidence of the possibility of natural application of fuzzy sets. Examples can be found in engineering requirements, software costs, quality of software, and risk assessment of software. (Pedrycz and Peters, 1997)

PROBABILITY: RISK FACTOR IMPACT:	very high	high	medium	low	very low
catastrophic	high	high	moderate	moderate	low
critical	high	high	moderate	low	none
marginal	moderate	moderate	low	none	none
negligible	moderate	low	low	none	none

Figure 12. Linguistic Terms in Assessing Software Development Effort

5.2.3 Evolutionary Computation (EC)

Evolutionary computation (EC) uses biological evolution as a source of inspiration to solve problems of adaptation by generating, analysing and changing a population of possible solutions. It includes genetic algorithms (GA), evolutionary computation, and evolutionary strategies that are biologically inspired methodologies aimed at global system and process optimization. (IEEE)

5.3 Conclusion for Algorithmic Approaches

BBNs is a good algorithm for measuring software engineering process. The advantage of BBN is that it will calculate the probability of each variable state regardless of the amount

of evidence. However, the lack of substantial hard evidence will be reflected in greater value uncertainty. Another algorithm is computational intelligence (CI). It has three main components neural networks (NNs), fuzzy systems (FS), and Evolutionary Computation (EC).

Personally, I think CI maybe the future for measuring software engineering process. Because it has an ability to view, learn and solve the situation like human and it can be used in deep learning. For instance, some metrics is hard to describe the exact the situation. We cannot just say that if and only if the software engineers reach the thresholds of some metrics, those engineers are hardworking while others are not, and the boundary of threshold is hard to define. But, fuzzy systems in CI can use linguistic words to instead of specific boundary. For example. In fuzzy systems, we can use fuzzy sets instead of exact value of metrics, like huge number of lines of code, low number of commits and medium productivity. Last, CI could be used to analyse the third level metrics for patterns as well. As learning and using patterns to figure out results are the initial form of deep learning, CI can combine software engineering with AI. I believe CI will be widely used to measure software engineering process one day.

6 Ethics

Collecting and measuring employees' data is very common in the companies. However, while these techniques of employee monitoring seem to improve productivity, what are the ethical problems with tracking the movements of a worker? Are businesses violating the privacy of their employees? William (2016) pointed out that *"Unlike safety, waste or fraud protections that are often specified in the employee interview or new hire orientation, an employee may feel that monitoring a badge for purposes other than these reasons may have too much of a ' Big Brother ' feeling."*

6.1 Determine the Ethics.

How to determine the ethicality of collecting and measuring data? William (2016) outlines four questions that helps to figure out this issue:

1. What information is the company gathering?
2. How will the information gathered be obtained?
3. How will the information gathered be used?
4. How transparent is the company with its employees about tracking?

In my view, employer who collecting the data must be responsible and reasonable. Only appropriate data should be collected by legal platforms. Companies also need to explain to workers the application of data tracking from them and get employees' permission. Moreover, Sandford-Brown (2014) mentioned that it is worth considering that there are also a number of ethical considerations that encourage employee monitoring, such as the need to avoid leaks of sensitive information, to stop infringements of company policies, to recover lost communications and, for instance, to limit legal liability.

6.2 Similar Steps for Ethical Compliance.

Similar steps for ethical compliance are also proposed by Sandford-Brown (2014):

1. **Set Written Policies**: Define a clear code of ethics that ensures that both employers and employees have a clear understanding of acceptable workplace behaviour.
2. **Inform Employees**: Make full disclosure of any stealth monitoring systems that are being implemented so that employees are fully aware that they are being monitored.
3. **Uphold Ethical Standards**: Ensure that the monitoring exercise remains moral, recognising that the employee still has a limited degree of privacy at work.
4. **Employee Participation**: Involving employees in the decision-making process on how the data are collected and how they are monitored will provide a common basis for establishing principles that are considered ethical by both employers and employees.

7 Conclusion

In conclusion, I agree that evaluating the software engineering process is difficult and the challenge is that there is no one single metric, platform or algorithm can directly gain the answer. But I believe software engineering process is measurable by the combination of multiple methods. Because there is plenty of data available to measure the software engineering process. Three levels metrics, numerous computational platforms like GitPrime, Velocity…and lots of algorithms like BBNs and CI can be combined to measure productivity. Moreover, we can figure out each drawbacks and advantages. In this case, using multiple metrics in multiple computational platforms and algorithms can make up for each method's shortcoming and measure aspects as much as possible in software engineering. At the same time, some new algorithms like CI, link software engineering to artificial intelligence to provide new ideas for measurement. However, we need to concern the ethics. Many software engineers care about whether the data collected by employers is in ethics, so employers should have the permission from employees, gain the data in legal and let use of data become transparent.

8 References

1. Fenton, N. E., and Martin, N. (1999) "Software metrics: successes, failures and new directions." Journal of Systems and Software 47.2 pp. 149-157.
2. Johnson, Philip M., et al. "Improving software development management through software project telemetry." IEEE software 22.4 (2005): 76-85.
3. Johnson, Philip M., and Hongbing Kou. "Automated recognition of test-driven development with Zorro." Agile Conference (AGILE), 2007. IEEE, 2007.
4. R. Want, A. Hopper, a. Veronica Falc and J. Gibbons. The active badge location system. ACM Trans. Inf. Syst., 10(1):91–102, 1992.

5. P.M. Johnson et al., "Beyond the Personal Software Process: Metrics Collection and Analysis for the Differently Disciplined," Proc. 25th Int'l Conf. Software Eng. (ICSE 03), IEEE CS, 2003, pp. 641–646.
6. 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.
7. G. C. Murphy, M. Kersten, and L. Findlater, "How are Java software developers using the Eclipse IDE?" IEEE Software, vol. 23, no. 4, pp. 76–83, Jul. 2006.
8. E. B. Passos, D. B. Medeiros, P. A. S. Neto and E. W. G. Clua, (2011) "Turning Real-World Software Development into a Game," Games and Digital Entertainment (SBGAMES), 2011 Brazilian Symposium on, Salvador, 2011, pp. 260-269.,
9. 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 research", 2010, ACM: Santa Fe, New Mexico, USA. p. 161-166.
10. Di Penta, M. "Mining developers' communication to assess software quality: Promises, challenges, perils." In Emerging Trends in Software Metrics (WETSoM), 2012 3rd International Workshop on. IEEE.
11. Yano, Kazuo et al. "Measuring Happiness Using Wearable Technology — Technology for Boosting Productivity in Knowledge Work and Service Businesses —." (2015).
12. Johnson, P.M., et al. (2003). Beyond the Personal Software Process: Metrics collection and analysis for the differently disciplined. 641- 646. 10.1109/ICSE.2003.1201249.
13. Toolkit, Leap and Moore, Carleton. (2000). Lessons Learned from Teaching Reflective Software Engineering using the Leap Toolkit.
14. P. Johnson, "Searching under the Streetlight for Useful Software Analytics" in IEEE Software, vol. 30, no. 04, pp. 57-63, 2013. doi: 10.1109/MS.2013.69
15. Bibi, S., Stamelos, I. and Angelis, L., 2003, November. Bayesian belief networks as a software productivity estimation tool. In *1st Balkan Conference in Informatics, Thessaloniki, Greece*.
16. Sousa, Abraham., et al, (2011). A Systematic Review of Bayesian Belief Networks in Management of Software Development Process.
17. De Sousa, A.L., Dertzbacher, J., Tierno, I.A., Birnfeld, K. and Nunes, D.J., A Systematic Review of Bayesian Belief Networks in Management of Software Development Process.
18. Pedrycz, W. and Peters, J.F., 1998. *Computational intelligence in software engineering* (Vol. 16). World Scientific.
19. What does Software Engineering mean?

    https://www.techopedia.com/definition/13296/software-engineering
20. We Can't Measure Anything in Software Development -- John Sonmez
    https://simpleprogrammer.com/we-cant-measure-anything-in-software-development/

21. Productivity Formula
https://simplicable.com/new/productivity-formula
22. Software engineering -- Computer Hope
https://www.computerhope.com/jargon/s/softengi.htm
23. Line of Code (LOC) Matric and Function Point Matric -- Ankush Singh
https://www.slideshare.net/diehardankush/loc-metric-and-function-point-metric/2
24. What Are Software Metrics and How Can You Track Them? -- Stackify
https://stackify.com/track-software-metrics/
25. Development Leaders Reveal the Best Metrics for Measuring Software Development Productivity -- Stackify
https://stackify.com/measuring-software-development-productivity/
26. Are Test Coverage Metrics Overrated? -- Tom Clement Oketch
https://www.thoughtworks.com/insights/blog/are-test-coverage-metrics-overrated
27. Software Engineering Metrics: The Advanced Guide -- Anaxi
https://anaxi.com/software-engineering-metrics-an-advanced-guide/
28. Is Test Coverage a Good Metric for Test or Code Quality? -- Chris Cooney
https://hackernoon.com/is-test-coverage-a-good-metric-for-test-or-code-quality-92fef332c871
29. What is Computational Intelligence? -- IEEE
https://cis.ieee.org/about/what-is-ci
30. When workplace monitoring, behavioural analytics, and employee privacy collide -- Terri Williams
https://www.digitalethics.org/essays/when-workplace-monitoring-behavioral-analytics-and-employee-privacy-collide