Measuring the Software Engineering Process

David Kilroy
Trinity College Dublin
kilroyda@tcd.ie

Abstract
The issue of measuring the productivity of a software engineer is one which has not had a miracle cure in the lifetime of the profession. Some metrics which were traditionally used are not adequate methods of measurement. This essay addresses this topic and analyses some models and algorithms which may be suitable alternatives to the traditional methods of measurement of the software engineer as well as the ethics and computational platforms available to carry out such data collection.

Introduction

Software is among the most complex structures created by professionals so why do we expect crude and simple metrics to be suitable ways of measuring an engineer's productivity? As described by Brooks in his paper *"No Silver Bullet - Essence and Accident*"[1] it's clear that most of this complexity is inherent in the system so it is not an option to make the software simpler to adhere to simpler measurement methods. Instead more detailed metrics must be measured to accurately capture the productivity of a software engineer. In this essay, I will briefly describe the methods used commonly in industry up until this point and then I will focus on a few specific methods that I have found to be interesting new ways in which we could measure the software engineering process more completely and accurately. It is important to note that I don't think any one of these methods alone will be a perfect way to measure the productivity of an engineer but I see promise in them as small parts of a potentially better method than measurement using other crude measurements we have seen in industry for the past few decades.

Traditional Software Metrics

The most common metric traditionally used to measure productivity is Lines of Code (or LOC) per unit of time [for instance KLOC per Week]. Despite continuous arguments against the use of this metric as a measure of software productivity, up until the 90's it was still commonly used in industry due to it being

easy to measure and visualise the size of the program. Computationally only a very simple line counting program would be required to obtain this metric which probably contributed towards it's continued use.
It is clear that motivation to use metrics in general by software engineers is (unsurprisingly) low so I believe any proposed model of software measurement must address this reality if it is to be used in industry.

Regardless, the measurement of this metric is clearly not suitable as a measure of engineer productivity. As famously attributed to Bill Gates:

> "Measuring programming progress by lines of code is like measuring aircraft building progress by weight."

Not only is it a bad measurement of actual problem coverage of a program, it can easily be manipulated by the engineer through churning or spacing out of code to make themselves seem more productive than they are. This issue of a measurement system being easy to manipulate must be addressed if it is expected to be an effective one.

Measurement of defect counts and McCabe's cyclomatic number are other metrics which are commonly measured in industry. Both of these measures are better metrics to methods of measurement than LOC as they take into account the problem being solved but they are far from complete models by themselves.

The cyclomatic complexity of a piece of code is measured by the amount of flow control statements present in it. A piece of source code with none of these statements, for example, has a complexity of 1. In industry, this has been used as a software metric to test code by having a threshold of complexity that the system must not exceed. The measurement of this metric is based off the assumption that more complex code has more errors. This assertion is challenged by Fenton & Neil (1999) where they strongly rejected the hypothesis that:

> "Modules with higher incidence of faults in all pre-release testing likely to have higher incidence of faults in post-release operation"

As complexity metrics aim to provide insight to the parts of a software system are more likely to have errors based off of their complexity, this study leaves them as an inadequate method of measurement of productivity by themselves.

For any such model to be a viable way of measuring a software engineers productivity, it must encompass the system as a whole (i.e. not just focus on one single area to measure) yet not so complex that it is an inconvenience to the people using it.
It’s no surprise that the measurement of any one simple metric is not an accurate representation of a developer’s productivity but the assertion that this means the software creation process is inherently too complex to measure (as can be seen in numerous media articles about the subject[2]) does not logically follow. We just need an equally complex measurement model that is up to the task.

Measuring Happiness, the 1/t fluctuation
As explained in their paper[4], Hitachi proposes a model of measuring productivity indirectly by measuring the physical activity of their subjects in combination with meeting KPIs (Key Performance Indicators). The results of their study found there to be a close relationship between the “trinity” of physical activity, happiness and productivity. As happier people are more likely to be more productive, they propose a model where happiness is measured indirectly by measuring a person’s body activity. This metric is clearly not a complete method of calculating an engineer’s productivity but I’m convinced it is an important one to measure nonetheless. The inclusion of KPI’s into the model in addition to the measurement of physical activity certainly makes this model a much more promising one as well. It is important to remember that as strange as this method initially sounds, they are not the only ones to see the correlation between employee happiness and productivity. The presence of nap rooms, gyms and gourmet food canteens in large companies such as Google[10] seems to suggest that they too recognise the correlation between employee wellbeing and productiveness.

How do they measure happiness computationally?

In order to collect the data required to measure physical activity, wearable sensors were worn by employees over a nine year period. From this data, they discovered a correlation between change in physical activity with the person's happiness which they call the "1/T fluctuation". After this data for physical activity was collected, they analysed how the employees were feeling based on a mix of questions about factors that relate to happiness such as *"concentration, enjoyment, desires, good sleep, conversation, appetite, depression, anxiety, loneliness, and sadness".* They found there to be a strong correlation between the 1/T fluctuation data and the mean values assigned to the questions asked in the questionnaires (or happiness of the employees). They in turn claim that this then has a positive effect on the productiveness of the employees being tested.

The trend of creating a particular task and making it into a game is prevalent in a number of areas today. With websites like Duolingo or Khan Academy, the user is rewarded for their progress towards learning a new grammar technique for a foreign language or watching an educational maths video. The idea of creating a program to reward developers for reaching milestones in a video game-like fashion is one which would no doubt increase enjoyment and competition in the workplace. If it is robust enough, the game system could be used to compare how engineers are performing compared to each other in terms of the programming goals they reach. While it is not a full measurement system in itself, when combined with the measurement of other metrics, the happiness and therefore the productivity of the entire system is bound to increase where motivation and competition levels are higher.

The second vitally important part of the study involved applying this measurement of happiness to the creation of better KPIs (which are used to measure the performance of an individual employee). Computationally, Hitachi used their artificial intelligence known as "H" to generate these KPIs taking into account the 1/T fluctuation of all the people who participated in the study. The use of AI to generate a large number of hypotheses and outcomes on a daily basis increased happiness and productivity in the study.

The ethical concerns I would have with this system of measurement mainly revolve around the collection of continuous data of physical activity. If this data is

not secured properly, this could potentially be ethically concerning. The use of KPIs to compare engineers to each other is inherently more inconvenient for the engineers but I do not believe it to be hugely concerning from an ethical point of view.
I believe the incorporation of an engineers happiness into the system of metrics (which includes the measurement of other metrics) combats the apathy towards measurement which is common in industry which could lead to this becoming a viable metric to measure in the software engineering process in practice as well as in theory.

Bayesian Belief Nets
Another model I have found that I believe to be a good method of measuring the software development process is to use Bayesian Belief Nets. This model, outlined in Fenton & Neil (1999)[6], seems to be the most practically viable of the ones I will address. A software system is modelled as a directed graphical network with each node representing uncertain an variable (such as testing quality, defects discovered or problem complexity) with an associated set of probability tables connecting the variables.

The edges pointing to any node in the graph signify that there is a combined conditional probability from the parent nodes to the child. The probability values are obtained by a mixture of empirically measured data and subjective judgements, which is the case in all BBNs. The modelling of probabilities from both the subjective and objective allows an organisation to view the whole software system and all the potential variables involved in making it in an intuitive graphical way. The modelling of all the variables involved with the software creation process allows for the managers of a software system to know what they must focus on and which areas their employees have been ignoring.

From a practical point of view, using this particular model allows the engineer to not have to explicitly model uncertainty values or the relationship of certain metrics to other ones. Provided with tools to be able to calculate probability tables and graph of the BBN, the use of this model would work well in industry as all the metrics in the network can be simply collected or easily provided by the engineers themselves.

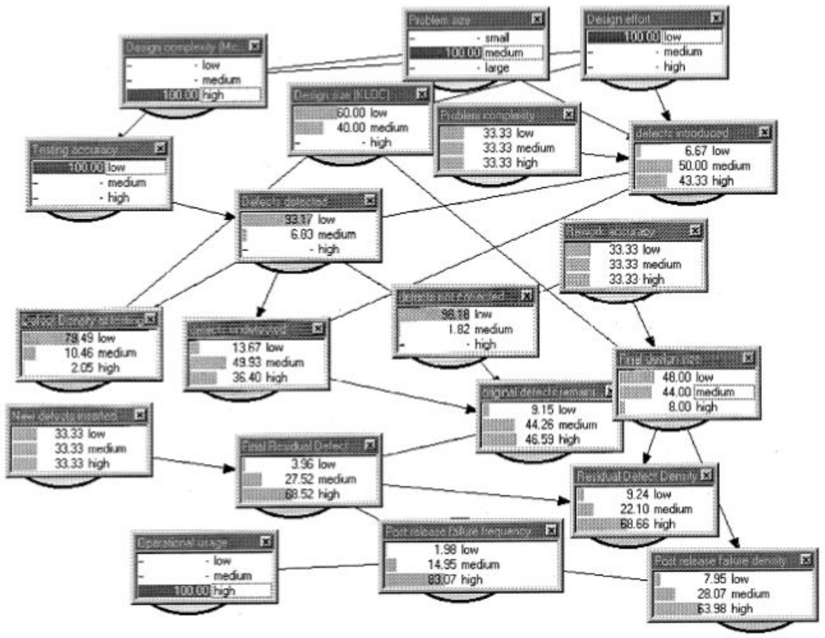

Fig 1. From Fenton & Neil (1999), Example BBN

Tools to Collect the Data

In general two main types of information are measured in software engineering: product metrics and process metrics. This data can tend to be quite hard to obtain due to a number of issues like errors in the human-collected data or the time it takes to incorporate such measurement methods into the system.

I have already addressed some potential methods to make sure this data is appropriate to measure and more enticing to collect but as well as these there are also computational tools available to collect this data.

Hackystat

Hackystat is a service-based framework for collecting, analysing and visualising all sorts of data related to the software engineering process. This data is collected by attaching software "sensors" to the engineers software tools which is then sent to the hackystat web service. Hackystat has looked into developing a game-structure like the one addressed earlier with developers gaining points based on how often they commit, the data trends of their programming process and so on. The idea of gamification is one which I think would be a great tool to increase worker productivity and the developers behind hackystat recognise this.

One particular aspect of the Hackstat service which is of particular ethical concern is the use of their Twitter functionality. Hackystat sends the software developments metrics in the form of a tweet with a linked account. The followers of this twitter account can then see these tweets and analyse the data. To me, this particular service seems to pose a security risk to the data which would need to be maintained securely.
I do not believe that there is an inherent ethical concern with the collection of this data to be used within a business as long as it is maintained securely and responsibly. The collection of metrics and employee data is a fundamental part of most if not all professional environments. From clock-in systems to administer correct wages to KPIs ensuring car sales are being made, performance metrics are an inevitable part of the working world so it would be unusual to assume software engineering industry should be any different.

However, the ethical concerns of collecting this data comes when it is not used for it's intended purpose. The recent measures taken by the EU (i.e. GDPR) to protect data from being misused are, *in theory*, a step in the right direction as how we as a society are aware that we need to take care of what happens to our data. (Although in practice, I don't have high hopes for the regulations beyond to annoy internet users with more popups). With great power comes great responsibility so more measures must be taken to makes sure this data is secure and will not be used irresponsibly.

PROM Automated Tool[12]
The PROM tool is a tool designed to collect both product and process data of software. In order to collect much of the data regarding the engineering process, the tool uses the hackystat framework to do so. The system was built to satisfy the conditions that: the architecture of the system is extensible to support future IDEs, types of data and analysis tools; the plug-in need to be as simple and the system needs to work offline. The system builds off of hackystat to store the data in their secured PROM database to allow clients the access the data and also delivers the data in an application to provide useful information for the developers and managers.

Conclusion

To gain relevant and useful data about the software engineering process, we can't use simplistic metrics that have been used previously to do so. The models I have found are steps in the right direction towards using methods in industry that are more suitable for collecting this data as they are more relevant in scope to the software engineering process. In order to implement these models, tools such as hackstat can be used to measure this data accurately but the security of this data must be of high priority if we are to make these complex methods ethical.

References

1. Brooks Jr. , Frederick P. - *No Silver Bullet - Essence and Accident in Software Engineering* (1986) - http://faculty.salisbury.edu/~xswang/Research/Papers/SERelated/no-silver-bullet.pdf

2. Fabulich, Dan - *You Can't Measure Software Engineering Productivity, so Measure Job Satisfaction Instead* (News Article, 2017) - https://redfin.engineering/measure-job-Satisfaction-instead-of-software-engineering-productivity-418779ce3451

3. Sonmez, John - *You can't measure anything in Software Development* - https://simpleprogrammer.com/we-cant-measure-anything-in-software-development/

4. Kazuo Yano, Dr. Eng. et. al. [of the Hitachi Group] - *Measuring Happiness Using Wearable Technology* [from Hitachi Review Vol. 64, No. 8] *(2015) -* http://www.hitachi.com/rev/pdf/2015/r2015_08_116.pdf

5. Sauermann, J., Lindquist, M. J., Zenou, Y. - *Network Effects on Worker Productivity (2015) -* https://www.researchgate.net/profile/Jan_Sauermann2/publication/285356496_Network_Effects_on_Worker_Productivity/links/565d91c508ae4988a7bc7397.pdf

6. Fenton, N. E., and Martin, N. (1999) - *Software metrics: successes, failures and new directions.* [from Journal of Systems and Software 47.2 pp. 149-157.] http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.63.2683&rep=rep1&type=pdf

7. W. Snipes, V. Augustine, A. R. Nair and E. Murphy-Hill - *Towards recognizing and rewarding efficient developer work patterns.* [from 2013 35th International Conference on Software Engineering (ICSE), San Francisco, CA, 2013, pp. 1277-1280.] https://dl.acm.org/citation.cfm?id=2486983

8. Johnson, Philip M., et al. - *Improving software development management through software project telemetry.* - https://pdfs.semanticscholar.org/5871/29969f8856c1dc5d233fcbaaa300e129f758.pdf

9. Passos, E. B. et. al. - *Turning Real-World Software Development into a Game* - http://www.sbgames.org/sbgames2011/proceedings/sbgames/papers/comp/full/30-91552_2.pdf

10. https://www.dailymail.co.uk/news/article-4665838/World-s-wackiest-workplace-look-inside-Google-offices.html (Couldn't find a more reputatable media source but proves the point adequately)

11. What is Cyclomatic Complexity? https://en.wikipedia.org/wiki/Cyclomatic_complexity#Explanation_in_terms_of_algebraic_topology

12. Silitti A., Janes A., Succi G., Vernazza T. *Collecting, Integrating and Analyzing Software Metrics and Personal Software Process Data* - http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.138.6806&rep=rep1&type=pdf