Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

CSU33012 Software Engineering

Measuring Software Engineering Report

Alice Doherty

Student Number: 19333356

Table of Contents

Introduction

What does it mean to be a software engineer? Despite what some may think, there is no right or wrong way to be an engineer but rather the process should be viewed as an art. When looking at software engineering in a professional capacity there are numerous reasons why a firm would want to measure the performance of their engineers. Some of these reasons include to track an individual or team's progress over time, to recognise the best (and worst) performers, as well as to help develop benchmarks and standards (Construx Software, 2016).

This report will explore how software engineering is currently being measured as well as the strengths and weaknesses of such practices. The first section of this report will look at what parts of software development we can measure, from processes to outputs, as well as the problems associated with these metrics. The second section will then explore the platforms that are available to individuals and enterprises to gather and process such data at scale. The next section will then look deeper into the algorithms used to calculate these metrics and perform analysis. Finally, the ethics of measuring software developers will be discussed, with particular focus on the ethics of using artificial intelligence (AI) to process huge amounts of individual data.

Measuring Engineering Activity

This section will look to explore *how* you can measure software engineering. The question at hand is not whether software engineering is measurable but rather what data is available to help measure engineering activity and whether the metrics being used are useful and fair. Although there are standards around software measurement, most notably the ISO Standard 15939 (International Organization for Standardization, 2017), there is no one definitive way to measure engineering productivity and this section aims to discuss some notable metrics and their strengths and weaknesses.

Source Lines of Code

A basic metric used to measure an engineer's output is to count the number of lines code they write. There are several ways to measure source lines of code (SLOC). One is to measure the number of actual lines in the source file (physical SLOC). Whereas more commonly, each statement is counted as a line of code (logical SLOC). Comments are generally excluded with only functional code being measured. A common criticism is that greater SLOC does not correspond to increased skill, and often when we refactor code, we look to reduce the amount of code. Two engineers can implement the same functionality but the SLOC could vary by a factor of ten (Construx Software, 2016). Additionally, SLOC is heavily dependent on the language and coding conventions used.

Commits & Pull Requests

The argument for measuring commits is that engineers are encouraged to make small and frequent commits, increasing transparency and incremental delivery. At its most basic, those who commit more should be rewarded. However, commits tell you nothing about value. Engineer A could be working on a challenging problem, making no commits for a week while they work out a solution, whereas Engineer B may be tasked with cleaning up comments in the code base making multiple commits a day. Who is adding more value?

Measuring pull requests (PRs) follows similar logic, with common criticisms being that it doesn't factor in the size or the effort of the work and may encourage small and unnecessary PRs. However, PRs can be useful to measure team performance. For example, by measuring the pull request lead time you can get an idea of how long it takes for a PR to get merged leading to better estimates. Additionally, PR discussions can be measured giving you an idea of how your team collaborates.

Function Points

Function points (FPs) attempt to measure engineering activity through the functionality offered to users and is code agnostic. This metric does not measure an individual engineer's productivity but rather is used to quantify the functionality offered by a project. Function point analysis (FPA) involves measuring the functional requirements in terms of *transactions* a user can perform and the *data* that can be stored/accessed with the software. Calculating

FPs involves identifying, classifying, and weighting each transaction or data type[1]. FPs are useful for quantifying an engineer's contribution to a business and are also not language specific so allow for more consistent comparison. However, they are complicated to calculate, and many managers view them as more effort than they're worth (Alvater, 2017).

Code Churn

As mentioned, quantity is not always a good metric for code quality. One measurement used to help evaluate the quality of engineering work is code churn. Code churn is when someone rewrites their own code within a short period of its first release (usually within three weeks). Although, it is normal for an engineer to make changes to their previous work as they go, high levels of code churn or unexpected fluctuations can signal to a manager that the engineer may be struggling. Like all metrics, the context needs to be considered, and what is interpreted as a worrying level of code churn on one project may be normal for another.

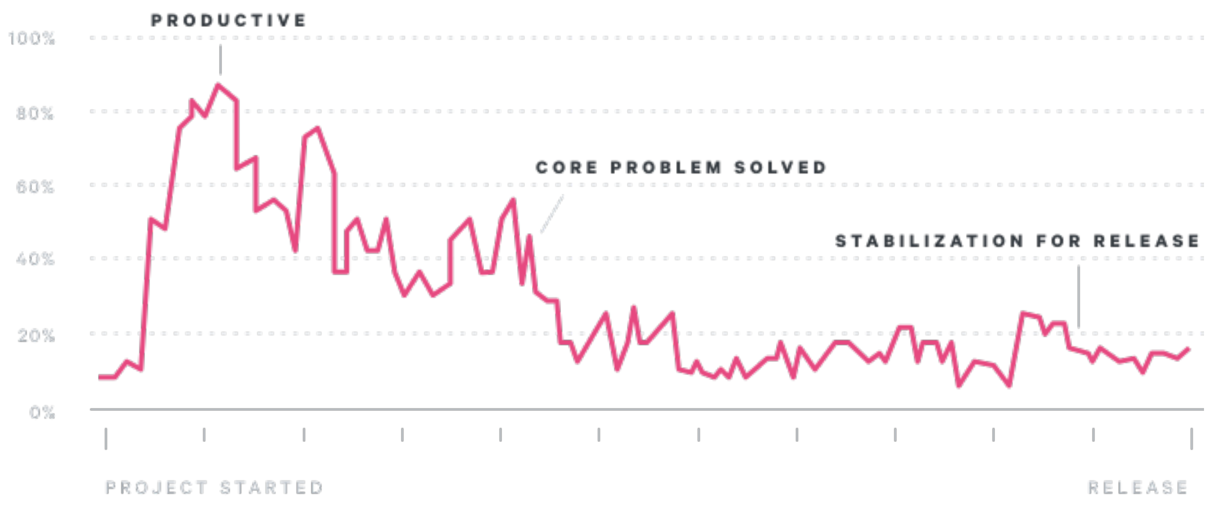


Figure 1: An example of "normal" fluctuation of code churn throughout a project's lifecycle. Note that the percentages of healthy code churn will vary from project to project and it is up to the manager to decide what is normal and how to interpret the levels. (Source: Pluralsight)

Some examples of what could cause a worrying level of code churn include, an engineer's lack of experience, or poor communication between stakeholders and ambiguous requirements. For example, Engineer A may commit 100 lines of code and Engineer B may commit 30. By measuring code churn it is revealed that Engineer A committed 40 lines of code in week one,

[1] For more detail on how to identify, classify and weight each functional component and carry out an analysis see the Total Metrics article from the reference section.

decided that they wanted to take a different approach and rewrote that code with 30 lines in week two, and then in week three decided to rewrite everything with 30 lines of different code. Although they may have committed 100 lines over the three weeks, Engineer A churned 100% of their code with none of their original code from weeks one or two adding any value to the project.

Agile Process Metrics – Team Velocity & Sprint Burndown

The next two metrics, team velocity and sprint burndown, are specific to measuring engineering activity when Agile methods are being used.

Team velocity measures the amount of work done by a team within a specific time, usually per sprint. It is usually measured in story points or hours. The benefits of this metric include more accurate delivery estimates, insight into whether the team is blocked by anything, and being able to quantify changes to results due to process changes (Infopulse, 2019).

Sprint burndown helps visualise a team's progress throughout a sprint. This is a useful way of measuring engineering activity as it allows you to visualise whether you are consistently finishing the work for the sprint early, or if you often miss sprint goals. This can then inform how the next sprint is planned so that the engineering is completed more efficiently.

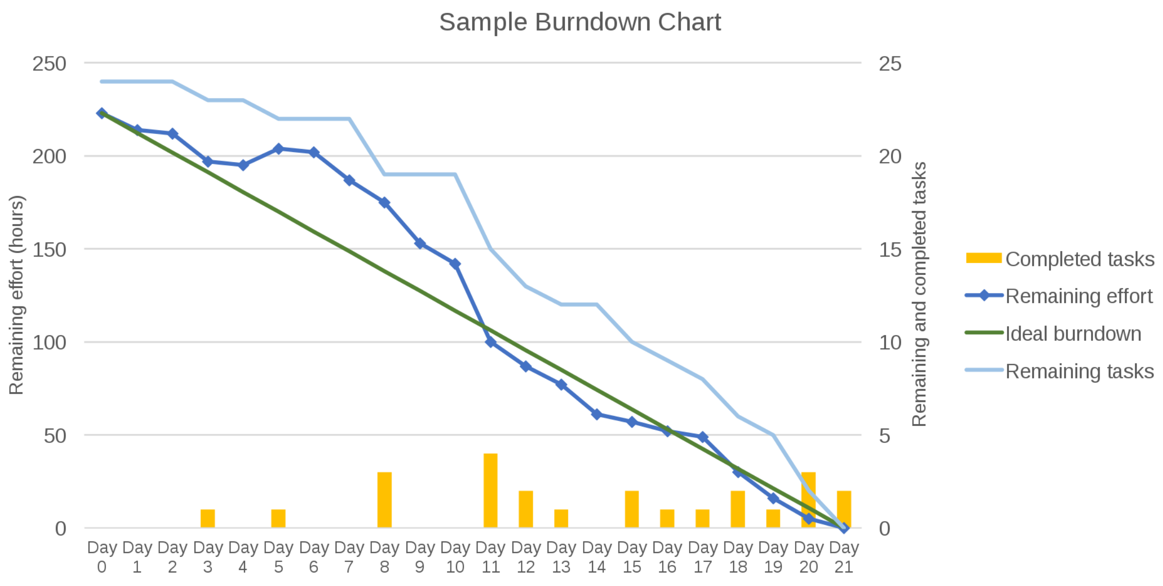


Figure 2: A sample burndown chart that plots a team's progress throughout a sprint. (Source: Wikipedia)

Problems with Measuring Engineering Activity

Measuring engineering activity is not straightforward. The metrics may be easy to measure but interpreting them usefully and fairly is complex. Abi Noda, previous Senior Product Manager at GitHub, gave an interesting talk[2] around this topic at GitHub Universe 2019 titled "The elusive quest to measure developer productivity" (Noda, 2019). Noda discusses what he calls the "Flawed Five", five measurements commonly used to measure software engineering that don't work, and that all share the characteristic of trying to measure output and productivity. The "Flawed Five" are:

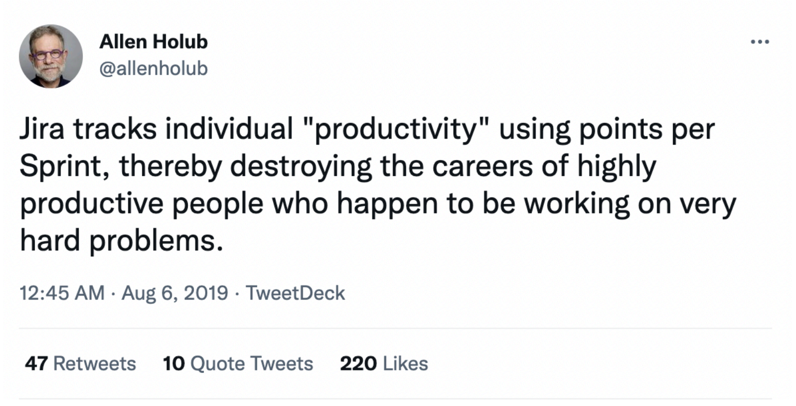
1. Commits
2. Lines of Code
3. Pull Request Count
4. Velocity Points
5. "Impact"

So, what are the overarching problems faced when measuring engineering activity? First, managers tend to collect and analyse data that is easily accessible, not necessarily metrics that are meaningful. Albert Einstein once said, "Not everything that can be counted… counts". Secondly, many of these metrics can be gamed, and therefore end up incentivising bad behaviours. For example, Noda describes the behaviour changes he noticed when average code review turnaround time was used as a metric for productivity. The obvious change was that the quality of reviews fell as people tried to finish as many reviews as possible. Additionally, code reviews were no longer being put up on Fridays as the software used to measure productivity did not take into account that people didn't work weekends. Another common criticism is that by taking these measurements it often disincentivises engineers to be creative or to take on more complex problems. For example, if your productivity is being measured by the number of pull requests, you will lean towards making smaller and safer pull requests.

Finally, the very crux of this problem is that it is very hard to define what "productivity" is, and it is context specific. Many simplify productivity down to output, but as we've already

[2] If interested, the talk itself can be viewed at https://www.youtube.com/watch?v=cRJZldsHS3c

seen, for example with lines of code, more is not always better. We have also seen that if we only look at output, engineers will game it, and this often leads to a decline in quality. I would take the view that software engineering needs to be viewed as an art, and not like a factory assembly line where you can easily measure what you are producing.



Allen Holub
@allenholub

Jira tracks individual "productivity" using points per Sprint, thereby destroying the careers of highly productive people who happen to be working on very hard problems.

12:45 AM · Aug 6, 2019 · TweetDeck

47 Retweets **10** Quote Tweets **220** Likes

Figure 3: A screenshot from Twitter that touches on the problem with measuring an individual's "productivity". (Source: Allen Holub, Twitter)

What is the solution? There is no one-size-fits-all solution and, as mentioned, I believe engineering should be viewed as an art. Therefore, it takes a good lead to define and interpret what it means to be productive, not a standalone metric. However, we can’t scrap these measurements altogether. Noda suggests what we should measure processes over outputs. Some metrics that measure processes include, code review turnaround time, pull request size, work in progress, and time to open. Additionally, we should measure against targets as opposed to absolutes. No two teams work the same, even within the same organisation, so each team needs to define their own goals. Finally, Noda advises against using individual metrics, as experience tells us that any benefits they introduce are never enough to counteract the disadvantages. However, this raises the question, if we shouldn’t measure individuals how do you identify the strongest and weakest members, especially as there is evidence that on average the strongest engineer is 20 times more productive than the weakest (Construx Software, 2016)?

Development Analysis Platforms

In the first section of this report, we looked at *what* we could measure. This section aims to explore *how* we can measure it in terms of the platforms available to gather and process this data efficiently.

Pluralsight Flow

Flow, originally GitPrime, is a platform used to measure software engineering productivity. It works by taking data available from Git such as pull requests, commits, and tickets to provide metrics about their software development process. Flow calculates a huge number of metrics[3] covering a wide range of the development process, from the codebase to a team's collaboration habits. To keep this concise I will only touch on some sample example metrics.

Flow provides a "code fundamentals" dashboard which concisely visualises a number of individual and team metrics. Managers can use this as a starting point for discussion during one-on-ones. One sample use case is using the impact measurement to gauge how long it takes for a new hire to create value for an organisation.

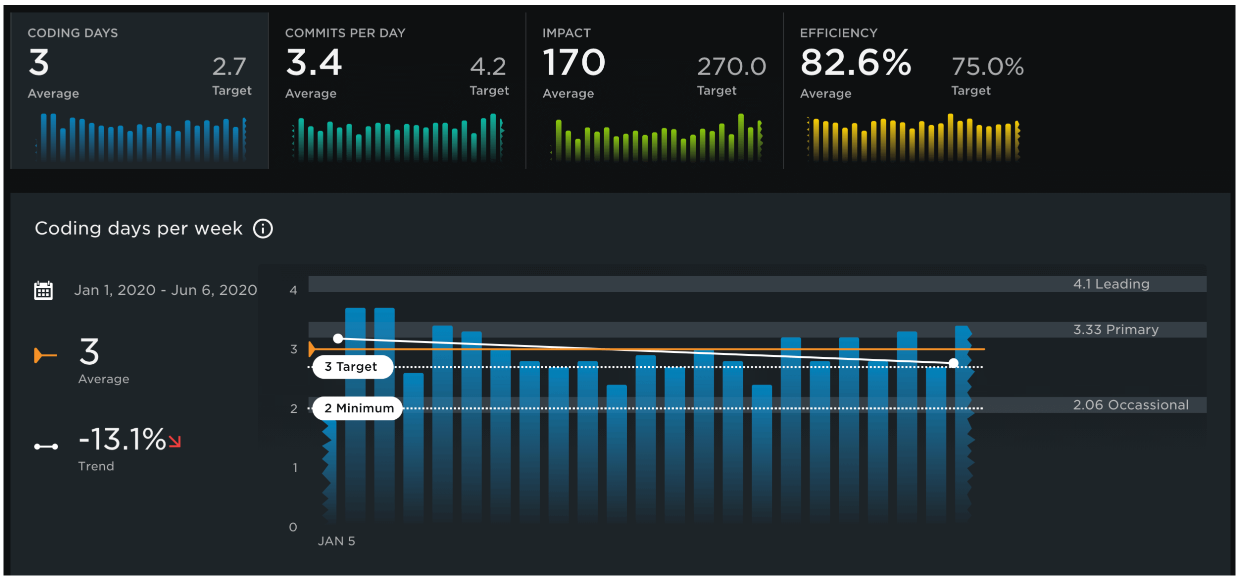


Figure 4: A sample of Flow's "code fundamentals" dashboard. Key metrics such as commits per day and impact are visualised. (Source: Pluralsight)

[3] A list of all the common metrics and what and why they are measured can be found here: https://help.pluralsight.com/help/metrics

Another sample dashboard offered by Flow is the “project timeline”. This visualises a team’s total output and allows you to see how much time is spent writing new code versus altering legacy code.

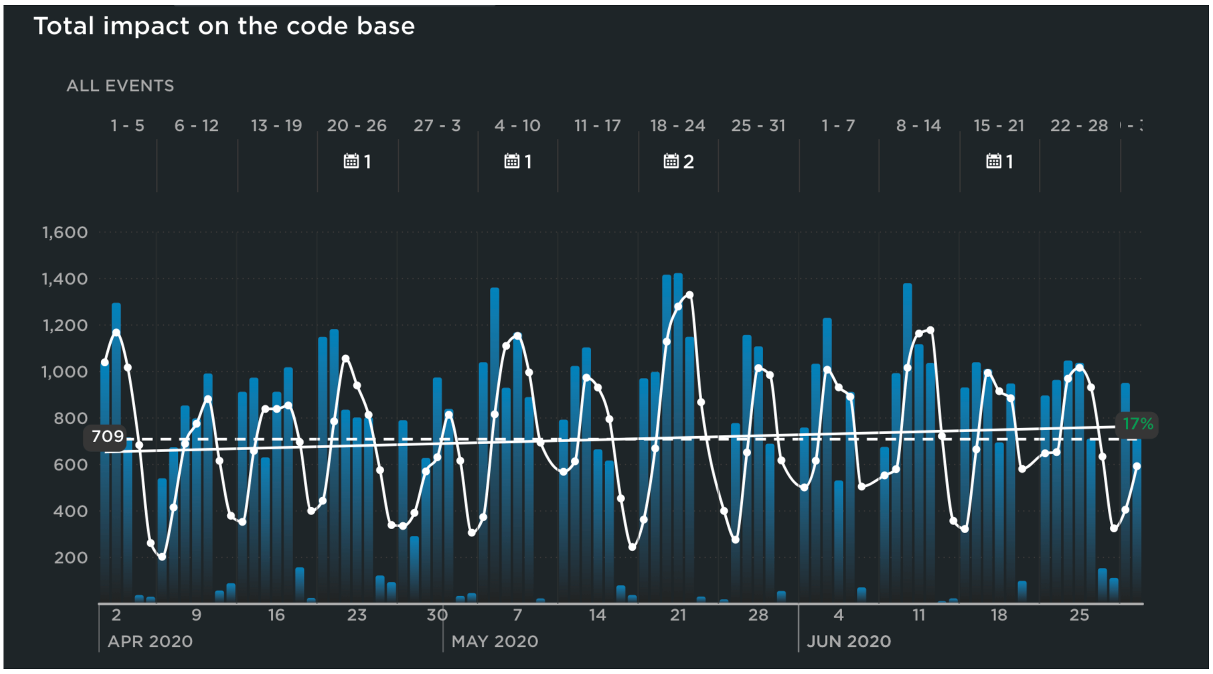


Figure 5: A sample of Flow's "project timeline" view. The total impact on a code base over a specified period is visualised. (Source: Pluralsight)

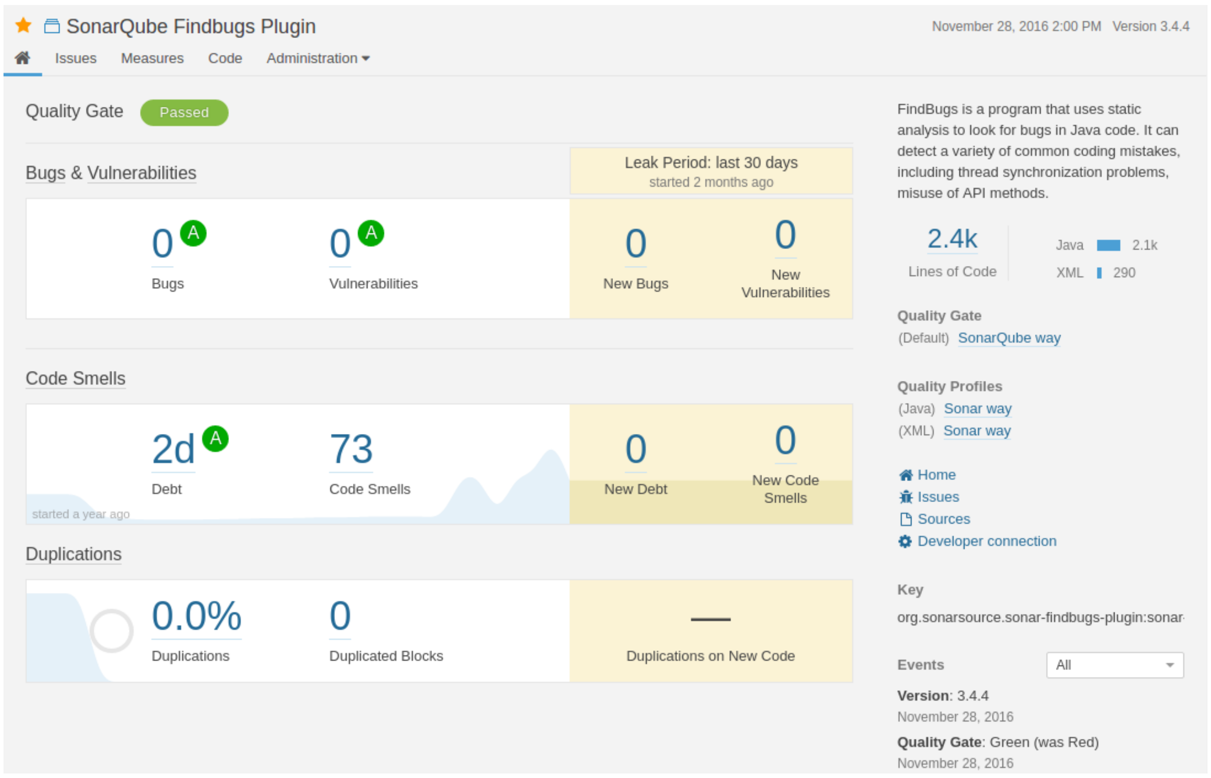
Other products similar to Pluralsight Flow include Waydev[4], LinearB[5], and Jellyfish[6].

SonarQube

Unlike Pluralsight Flow which leans towards measuring developer productivity, SonarQube is an open source platform used to help measure code quality and security. Their products range from a free community edition to paid editions for developers and enterprises. It is used to analyse code and detect bugs, code smells, as well as security vulnerabilities. It also compiles reports about the code in regards to a number of issues such as adherence to coding standards, code coverage and testing, code complexity, and commenting. Unlike Pluralsight Flow which retrieves data from Git, SonarQube performs static analysis of code and can be

[4] See https://waydev.co/
[5] See https://linearb.io/
[6] See https://jellyfish.co/

integrated with things like Maven and Gradle, or integrated with IDEs like Visual Studio and IntelliJ.


SonarQube Findbugs Plugin

November 28, 2016 2:00 PM Version 3.4.4

Issues Measures Code Administration

Quality Gate Passed

Bugs & Vulnerabilities

Leak Period: last 30 days
started 2 months ago

0 A	0 A	0	0
Bugs	Vulnerabilities	New Bugs	New Vulnerabilities

Code Smells

2d A	73	0	0
Debt	Code Smells	New Debt	New Code Smells

started a year ago

Duplications

0.0%	0	—
Duplications	Duplicated Blocks	Duplications on New Code

FindBugs is a program that uses static analysis to look for bugs in Java code. It can detect a variety of common coding mistakes, including thread synchronization problems, misuse of API methods.

2.4k Lines of Code

Java	2.1k
XML	290

Quality Gate
(Default) SonarQube way

Quality Profiles
(Java) Sonar way
(XML) Sonar way

Home
Issues
Sources
Developer connection

Key
org.sonarsource.sonar-findbugs-plugin:sonar

Events All

Version: 3.4.4
November 28, 2016

Quality Gate: Green (was Red)
November 28, 2016

Figure 6: A sample landing page in SonarQube which shows the results of the FindBugs plugin which analyses Java code for bugs. (Source: Wikipedia)

Jira

Jira is a product developed by Atlassian and is one of the industry leaders when it comes to product management in software development. At a high level, Jira can be used for planning and tracking software development and by Agile teams to maintain scrum and kanban boards. As Jira is commonly used to organise workloads into sprints, it can easily extract data from its platform to calculate sprint/epic burndown, velocity, and control charts, to name a few examples (Radigan, n.d).

As sprint burndown and velocity were covered previously in this report, as an example I will focus on control charts. Control charts visualise the time for an issue to move from "in progress" to "done". This data is extracted from the data Jira has from the team's scrum or kanban board. In this way, it doesn't extract data from an external source like Git, but simply visualises what is already available on Jira so the lead/manager can interpret it.

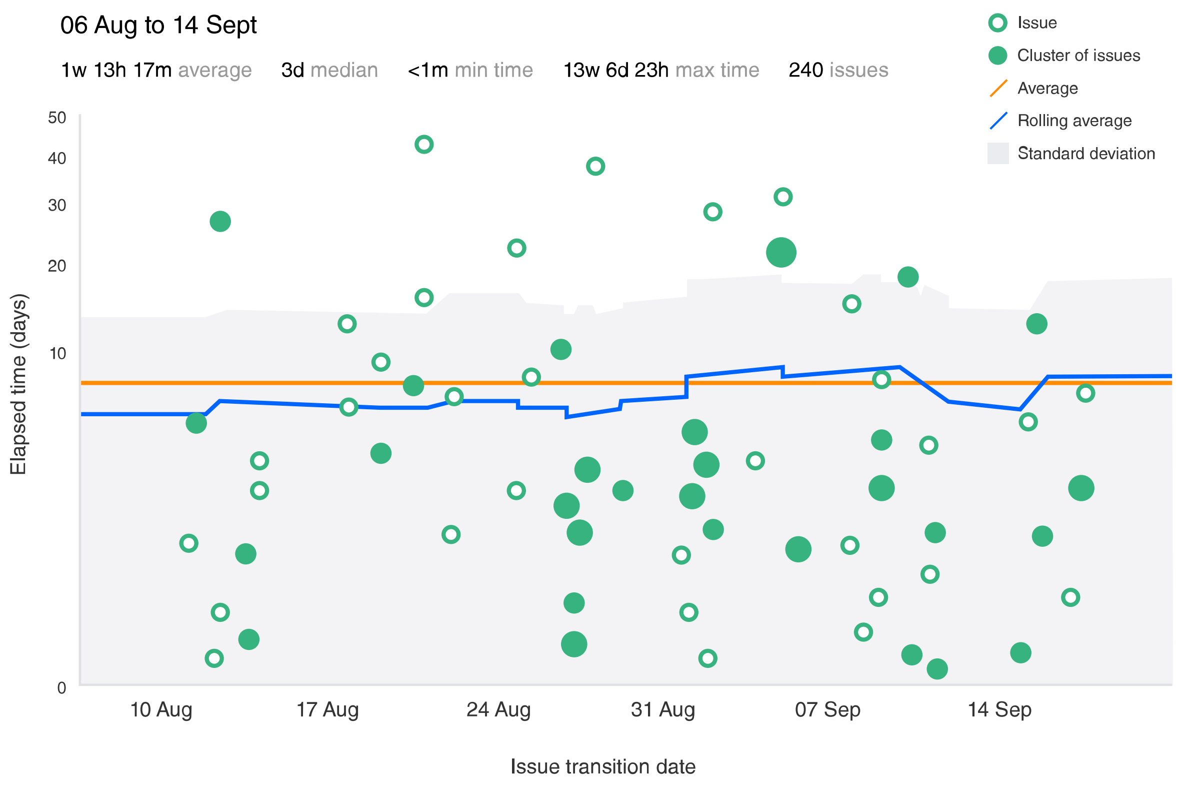


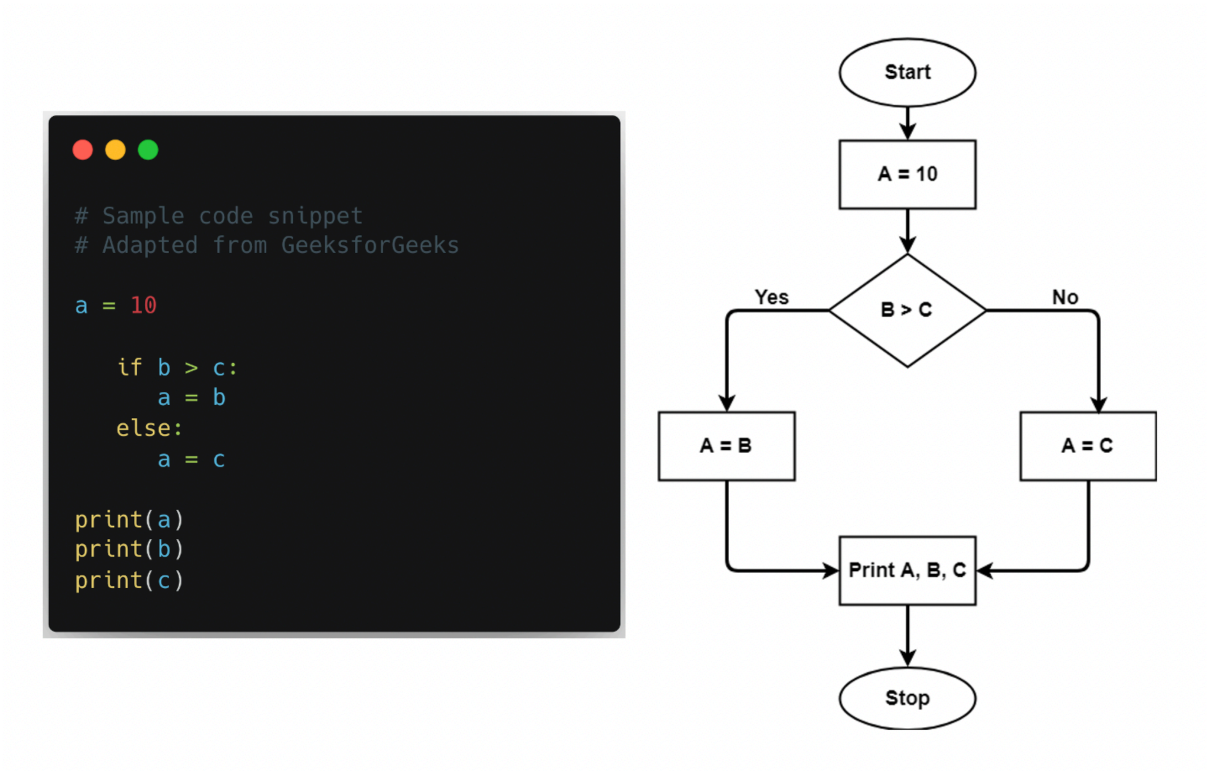
Figure 7: A sample control chart produced by Jira. (Source: Atlassian Agile Coach)

Algorithmic Approaches

Judging developers by using easily attainable metrics is clearly not enough to give a comprehensive overview of their performance. As illustrated by looking at the platforms above, to make more interesting analyses, sophisticated algorithms and AI is often used. This section will discuss some algorithms used to measure different aspects of the software development process and then discuss some of the shortcomings of these approaches.

Software Complexity – Cyclomatic Complexity

Unlike metrics such as the number of commits, quantifying a piece of software's complexity is not as simple, and there is not one objective value. A common measure of software complexity is McCabe's cyclomatic complexity metric. Cyclomatic complexity is defined as "the number of linearly independent paths" in a piece of code (GeeksforGeeks, 2021). The control flow graph of a program is used to compute its complexity. Each node in the graph represents an indivisible group of commands, and there is a directed edge between two nodes if the second command can be executed directly after the first. For example, if the code contains one if-statement, the cyclomatic complexity is 2 because there are two possible paths, true or false. Figure 8 shows an example code snippet and how that is translated into a control flow graph.



```python
# Sample code snippet
# Adapted from GeeksforGeeks

a = 10

   if b > c:
      a = b
   else:
      a = c

print(a)
print(b)
print(c)
```

Figure 8: An example code snippet and its corresponding control flow graph. (Source: GeeksforGeeks)

The equation used to calculate cyclomatic complexity (CYC) is $CYC = E - N + 2P$, where E is the number of edges in the graph, N is the number of nodes in the graph, and P is the number of connected components. So for the above example the complexity would be $7 - 7 + 2 = 2$.

Again, like all other metrics discussed, cyclomatic complexity does not offer an objective metric to judge an engineer on. For example, the complexity of an engineer's commit will depend on the task they were assigned in the first place, and so context must always be taken into account to avoid unfair comparisons.

Developer Productivity

It is not easy to quantify productivity as it is largely contextual. Products like Pluralsight Flow attempt to do this through various metrics but the exact details of such algorithms are not public and involve complex AI. In this section, I will discuss some more simplistic algorithms used to quantify a developer's productivity (Chatterjee, n.d). Often, the following equation is used to measure productivity: $Productivity = \frac{ELOC}{PM}$, where ESLOC is the effective/equivalent source lines of code, and PM is person months. When all code committed is new it is simple to calculate the SLOC (explained at the start of this report), however, if old code is modified other factors need to be taken into account. For example, how much time the developer spent understanding the legacy code to make their changes and if there was poor documentation the developer would need to spend time reverse engineering the code. An adaptation adjustment factor (AAF) is calculated using $AAF = 0.4F_{des} + 0.3F_{imp} + 0.3F_{test}$. F_{des} is the percentage of reused software that needed redesigning/reverse engineering, F_{imp} is the percentage of software code that needed to be modified/recoded, and F_{test} is the percentage of reused software that requires regression testing. Using the AAF, we now have $ESLOC = S_{new} + S_{mod} + S_{reused}$, where S is the number of new, modified, and reused lines of code respectively. However, how this final productivity figure is interpreted is down to the team itself, as a good figure for one team could be terrible for another. For context, Chatterjee writes that a team at Borland were producing 1000 lines/week, whereas a Microsoft Windows team were producing 1000 lines/year. The company Quantitative Software Management (QSM) collects SLOC figures from a range of products across different industries as well provides performance benchmark tables to clients. Additionally, SonarQube, one of the platforms mentioned previously, has a built in feature that calculates SLOC automatically.

Shortcomings

A manager can calculate all the metrics they want, or view AI generated reports but at the end of their day these are ultimately tools used to help their judgement. These algorithms alone should never be left to make the decisions themselves as we have not gotten to a stage where human judgement can be replaced. One could also argue that a lot of machine learning algorithms smooth out the noise and present an averaged out picture, however, at times those intricacies not caught by the system are crucial. On the other hand, one could argue that managers and leaders themselves often present biases and by using AI and other algorithms to calculate a variety of performance metrics it can help engineers be viewed on a more level playing field. More discussion on whether we should be measuring software development in these ways will be in the final section.

Ethics

We've discussed at length how software engineers are being measured, and the strengths and weaknesses of various metrics. Now it is time to ask whether we should be taking these measurements in the first place and the ethics behind this.

Effect on the Individual

Most firms employ these metrics and AI models to get the most of their employees, however, what effect does this have on the individual? We have already discussed that when individuals know they are being monitored they will change their behaviour, this is also known as the Hawthorne effect. The metrics we choose determine how an employee behaves, and oftentimes gaming these metrics leads to an individual changing their workflow, for the worse. This can become a dilemma when it comes to employees choosing between keeping their job and "playing the game", or taking on more creative and difficult projects that don't measure well against certain metrics. Is it fair to expect all members of a team to be compared against the same metrics? In my opinion, I think when used in moderation and to compliment a manager's judgement it can be useful. However, I have issue with the fact that by increasing the influence of these metrics on an individual's job, it is almost whittling down each position to a certain "ideal" and that individual quirks and personalities begin to get ignored and discouraged.

On the other hand, one could argue that through using these data driven metrics it could make the workplace fairer. It is no secret that many managers have their own internal biases, and oftentimes look to hire or promote those who are like them. By employing metrics based on hard data, we could reduce these biases and have an objective footing to measure employees against (this is assuming the metrics used are fair and accurate, which will be covered next). AI used on data collected on an individual’s development process could also help inform what projects individuals are most suited for.

However, what about when it comes down to firing or promoting someone over some figure these algorithms generate? We need to remember that the use of these could impact someone’s entire livelihood. This is not a space to be experimenting and playing around with. Is it fair for someone who is tasked with “quick fixes” to be measured against someone working on a larger project? Again, I think there is a healthy middle ground to be met in this situation. These measurements should all take context into account and not replace a manager’s judgement or be solely used to justify the firing or hiring of someone. In my opinion, these metrics are good to ensure an individual is within certain boundaries. For example, if someone’s number of commits fall in either of the two extreme tail points of the normal distribution it’s a good indicator to the manager to investigate their performance. But, for the people who lie in the middle I don’t think you should be comparing these people down to the figure.

Accuracy

A huge issue when it comes to the ethics of using these systems to measure people is their accuracy. In my opinion, as long as the algorithms are proven to be inaccurate or biased, they are unethical to use. Machine learning systems are only as good as the datasets they are trained on. One example of this is the gender biases in AI algorithms. The reality is that we don’t have as much sample data on women, especially in software engineering, as men. A famous example of this is the Amazon AI recruiting tool. The system itself was trained by looking at the CVs submitted to Amazon in the last decade, and most of these were those of men. Therefore, when used to automate the recruitment process, it favoured male applicants as the traits seen in their CVs were mirrored in the dataset the system was trained to view as good. In short, machine learning does not eradicate our human biases and often reflects them, and historical and social inequalities (Manyika, Silberg and Presten, 2019). Judging human behaviour is a subtle art and often the use of AI removes the intricacies associated with each individual. Products that produce an AI generated “productivity score” don’t reflect these and end up reducing everyone down to the same ideal. If an AI learns how to measure employee behaviour from a predominantly male sample, which is almost always the case when it comes to software development, females may be branded as less competent because they approach their job differently, but not less effectively. It is worth noting here that I am using gender as a basis for bias, but the same holds true for other minorities.

Use at a Team Level

In my opinion, when it comes to using AI to make judgements at an individual level, it is generally ethically questionable. However, I think it becomes more reasonable when you look at using AI to measure performance at a team or organisational level, without singling out individual people. For one, it is less of a privacy issue as you are not analysing data attributed to one individual, and additionally you are not having as much of an impact on someone's life. It is not reasonable to say we should never use these metrics, as we have always measured performance in one form or another and you would not know what was happening if you didn't. The difference now is the scale of data we have. Using AI to predict development patterns across a team can lead to better success predicters and estimates for stakeholders. You can also process the data a lot more accurately when you are looking at something less abstract such as output (rather than productivity).

Going Further than Software Development

Up until this point we have talked about metrics related to the software development process, however, a huge shift is being seen at the moment across industries when it comes to using AI to measure employee productivity. In my opinion, it is these products that are of most concern ethically. The onset of the Covid-19 pandemic catalysed the shift to remote working, and a side effect of this was an increase in surveillance software. One example is Hubstaff which records an employees' keyboard and mouse use, as well as the websites they visit. Another product is Time Doctor which can take videos of an employee's screen, as well as take a picture through their webcam every 10 minutes to check if they are still at their desk. From an ethical standpoint, I am completely against this as I view it as a complete invasion of privacy as well as a practice that ultimately undermines trust between employers and employees. My stand point is, if you can get the tasks you are assigned done on time it should not matter how you spend your day. Additionally, the knowledge that you are being monitored will change the way you work, and cause unnecessary anxiety.

Enaible is another such product and is an "AI that dynamically learns how you work, unlocking the fullest potential of your work, inspiring like no other in the world" (Enaible, n.d). It learns an individual's workflow, for example by learning what triggers them (emails, calls, etc.) and how long they spend on tasks. It then generates a productivity score, which is supposedly agnostic, meaning it can be used to compare employees across an organisation regardless of their job. To me this is a huge issue as employers will be tempted to compare all employees with this same score, despite each employee serving a different purpose, and undoubtedly creating value in different ways. This is also an issue for those who don't do repetitive work, and again going back to treating software engineering as an art, how can we boil down their work to a productivity score when their work is ever changing and contextual. My biggest ethical issue with this product is that they also have an algorithm that recommends how employees can improve. My fear with this is that employers will start seeing this as a call to optimise everything, including their workforce, leading to many losing their jobs. This shift towards optimising every process seems to be leading the way towards forgetting about employee well-being and focusing on what they output rather than the processes that get them there.

A Framework for Ethics of Data and AI

To finish, I would like to introduce a framework proposed by Josh Bersin in which he describes "Four Dimensions of Trust" (Bersin, 2019). On one axis, it asks whether the data and algorithms being used are fair. Does it accurately reflect the individual/team you are assessing and is it taking biases into account? On the second axis, it looks at whether the data systems and algorithms are safe. Are people's privacy being respected and is their data secure?

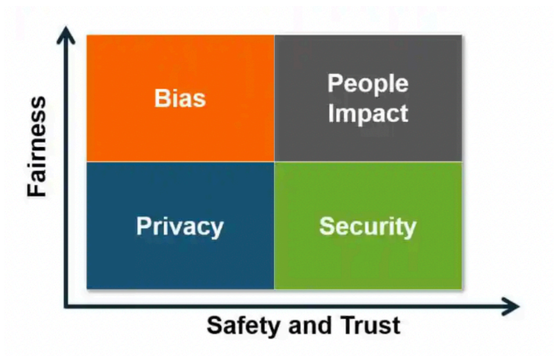


Figure 9: A framework for ethics of data and AI. (Source: joshbersin.com)

All four dimensions of this framework are essential to take into account when we adopt these new systems to measure our workforce, and it is a far from easy task.

Conclusion

The workplace is an everchanging, and ever competitive landscape and we will never escape the desire to be able to quantify an individual's value. Research looking into the correlation between an engineer's perceived effort versus an observable metric, such as those collected by Git, shows that they offer a good initial approximation of an individual's productivity but that even the most strongly correlated metric (line impact) only has 61% correlation with effort (Harding, 2021). However, there are many benefits to taking these measurements. First, for the individual, the platforms discussed in this report can easily identify developer's that are struggling by looking at those who fall way below the average. At a team level, these metrics offer invaluable insights into how a team is progressing and what processes do or don't work for them. At the end of the day, the best we can do is hold organisations accountable and explore in depth what they are measuring and doing with our data, and only then we can decide whether it is fair and accurate.

This report has explored in depth how software engineers are being measure, and in turn given us a better idea of what it means to be software engineer. We have discussed what data can be collected on a developer and the platforms used to process this efficiently. Additionally, the algorithms used to quantify more abstract concepts like complexity and productivity were explored and the ethical concerns around collecting and processing individual data.

References

Altvater, A., 2017. *What Are Software Metrics and How Can You Track Them?*. [online] Stackify. Available at: <https://stackify.com/track-software-metrics/> [Accessed 27 December 2021].

Bersin, J., 2019. *People Analytics and AI in the Workplace: Four Dimensions of Trust*. [online] joshbersin.com. Available at: <https://joshbersin.com/2019/05/the-ethics-of-ai-and-people-analytics-four-dimensions-of-trust/> [Accessed 30 December 2021].

Construx Software, 2016. *Measuring Software Development Productivity | Steve McConnell*. [video] Available at: <https://www.youtube.com/watch?v=x4IboMnTdSA> [Accessed 27 December 2021].

Chatterjee, D., n.d. Measuring Software Team Productivity. *Sutardja Center for Entrepreneurship & Technology, Berkeley Engineering*, pp.5-9.

Enaible n.d. [online] Available at: <https://enaible.io/> [Accessed 30 December 2021].

GeeksforGeeks, 2021. *Cyclomatic Complexity*. [online] Available at: <https://www.geeksforgeeks.org/cyclomatic-complexity/> [Accessed 28 December 2021].

Harding, B., 2021. *Measuring Developer Productivity: A Comprehensive Guide for the Data Driven - GitClear*. [online] Gitclear.com. Available at: <https://www.gitclear.com/measuring_developer_productivity_a_comprehensive_guide_for_the_data_driven> [Accessed 31 December 2021].

Heaven, W., 2020. *This startup is using AI to give workers a "productivity score"*. [online] MIT Technology Review. Available at: <https://www.technologyreview.com/2020/06/04/1002671/startup-ai-workers-productivity-score-bias-machine-learning-business-covid/> [Accessed 30 December 2021].

Infopulse, 2019. *Top 10 Software Development Metrics to Measure Productivity*. [online] Medium. Available at: <https://medium.com/@infopulseglobal_9037/top-10-software-development-metrics-to-measure-productivity-bcc9051c4615> [Accessed 27 December 2021].

International Organization for Standardization, 2017. *ISO/IEC/IEEE 15939:2017 Systems and software engineering — Measurement process*. [online] Available at: <https://www.iso.org/standard/71197.html> [Accessed 27 December 2021].

Pluralsight. n.d. *Introduction to Code Churn: Causes & Fixes*. [online] Available at: <https://www.pluralsight.com/blog/tutorials/code-churn> [Accessed 27 December 2021].

Manyika, J., Silberg, J. and Presten, B., 2019. *What Do We Do About the Biases in AI?*. [online] Harvard Business Review. Available at: <https://hbr.org/2019/10/what-do-we-do-about-the-biases-in-ai> [Accessed 30 December 2021].

Noda, A., 2019. *The elusive quest to measure developer productivity - GitHub Universe 2019*. [video] Available at: <https://www.youtube.com/watch?v=cRJZldsHS3c> [Accessed 27 December 2021].

Pluralsight.com. n.d. *Pluralsight Flow*. [online] Available at: <https://www.pluralsight.com/product/flow> [Accessed 28 December 2021].

Radigan, D., n.d. *Five agile metrics you won't hate | Atlassian*. [online] Atlassian. Available at: <https://www.atlassian.com/agile/project-management/metrics> [Accessed 28 December 2021].

Reisenwitz, C., 2021. *How to Measure Productivity in Software Engineering*. [online] Clockwise Blog. Available at: <https://www.getclockwise.com/blog/measure-productivity-development> [Accessed 27 December 2021].

Sonarqube.org. n.d. *SonarQube*. [online] Available at: <https://www.sonarqube.org/> [Accessed 28 December 2021].

Total Metrics. 2021. *What are Function Points?*. [online] Available at: <https://www.totalmetrics.com/function-point-resources/what-are-function-points> [Accessed 27 December 2021].