**Cover Page**

**Table of Contents**

**Introduction**

First, we must ask the question of why we want to measure software engineering. Steve McConnell (Construx Software, 2016) (see also <https://www.getclockwise.com/blog/measure-productivity-development>) gives a number of reasons why firms would want to measure the productivity of their engineers, including, to track their progress over time, to recognise the top engineers and, to help develop benchmarks and standards.

**Software Productivity**

<https://ifs.host.cs.st-andrews.ac.uk/Books/SE9/Web/Planning/productivity.html>

<https://medium.com/@infopulseglobal_9037/top-10-software-development-metrics-to-measure-productivity-bcc9051c4615>

Two types of metrics

* Size-related metrics
* Function-related metrics

Productivity metrics should be

* Consistent
* Auditable
* Available
* Repeatable

**Measuring Engineering Activity**

The first section of this report 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). For example, do we measure the number of actual lines in the source file (physical SLOC), or should we consider each statement to be a line of code (logical SLOC). In general, the latter is used, and comments are generally excluded with only functional code being measured. A common criticism is that greater SLOC does not necessarily correspond to increased efficiency or skill, and often when we refactor code, we look to reduce the amount of code. Two engineers can implement the same functionality but the number of SLOC could vary by a factor of ten (Construx Software, 2016). Additionally, depending on the language and coding conventions used the SLOC differs.

**Commits & Pull Requests**

Like SLOC, the number of commits and pull requests are an easy measurement to extract. 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, and vice versa. 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 to the product?

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/closed leading to better estimates. Additionally, PR discussions can be measured giving you an idea of how your team collaborates.

**Functions Point**

Whereas SLOC measures engineering activity from a code point of view, 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 an engineering project. Function Point Analysis (FPA) involves measuring the functional requirements in terms of *transactions* a user can perform with the software and the *data* that can be stored/accessed by the software. Calculating FPs involves identifying, classifying, and weighting each transaction or data type[[1]](#footnote-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 across the board. 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 called code churn. Code churn is when someone rewrites their own code within a short period of its first release (usually within three weeks). It is usually measured in lines of code or a percentage. Although, it’s 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.

Chart

Description automatically generated

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, 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.

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.

**Chart, line chart

Description automatically generated**

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

**Problems with Measuring Engineering Activity**

As previously touched on, measuring engineering activity is not straightforward. The metrics themselves may be easy to measure but interpreting them in a useful and fair is complex. Abi Noda, previous Senior Product Manager at GitHub, gave an interesting talk[[2]](#footnote-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. However, according to him they don’t work, and all share the common 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”

The downfalls of the first three have been discussed above and velocity points is similar to team velocity discussed under Agile process metrics. Impact is an abstract concept that is essentially lines of code but also considers how many files were touched, and how much new code was written.

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 measure value creation. There is an Albert Einstein quotation that summarises this flaw, “Not everything that can be counted… counts”. This leads into the second point, 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 code 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.

Graphical user interface, text, application, email

Description automatically generated

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

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 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.

We have now heard the criticisms, so what is the solution? There is no perfect one-size-fits-all solutions 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 may 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 and services available to gather and process this data efficiently.

**Pluralsight Flow**

Flow, originally called GitPrime and acquired by Pluralsight in 2019, is a platform used to help measure software engineering productivity. It works by taking data available from Git such as pull requests, commits and tickets to provide metrics to engineers and managers about their software development process. Flow calculates a huge number of metrics[[3]](#footnote-3) covering a wide range of the development process, from metrics about the codebase to metrics about a team’s collaboration. To keep this concise I will only touch on some sample example metrics, but this is in no way exhaustive.

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.

Graphical user interface

Description automatically generated with medium confidence

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

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.

Chart, histogram

Description automatically generated

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 which I don’t have space to discuss include Waydev[[4]](#footnote-4), LinearB[[5]](#footnote-5), and Jellyfish[[6]](#footnote-6).

**SonarQube**

SonarQube is another platform used to help measure software engineering activity. 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 integrated with things like Maven and Gradle, or integrated with IDEs like Visual Studio and IntelliJ.

Graphical user interface, application, website

Description automatically generated

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 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 progess” 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.

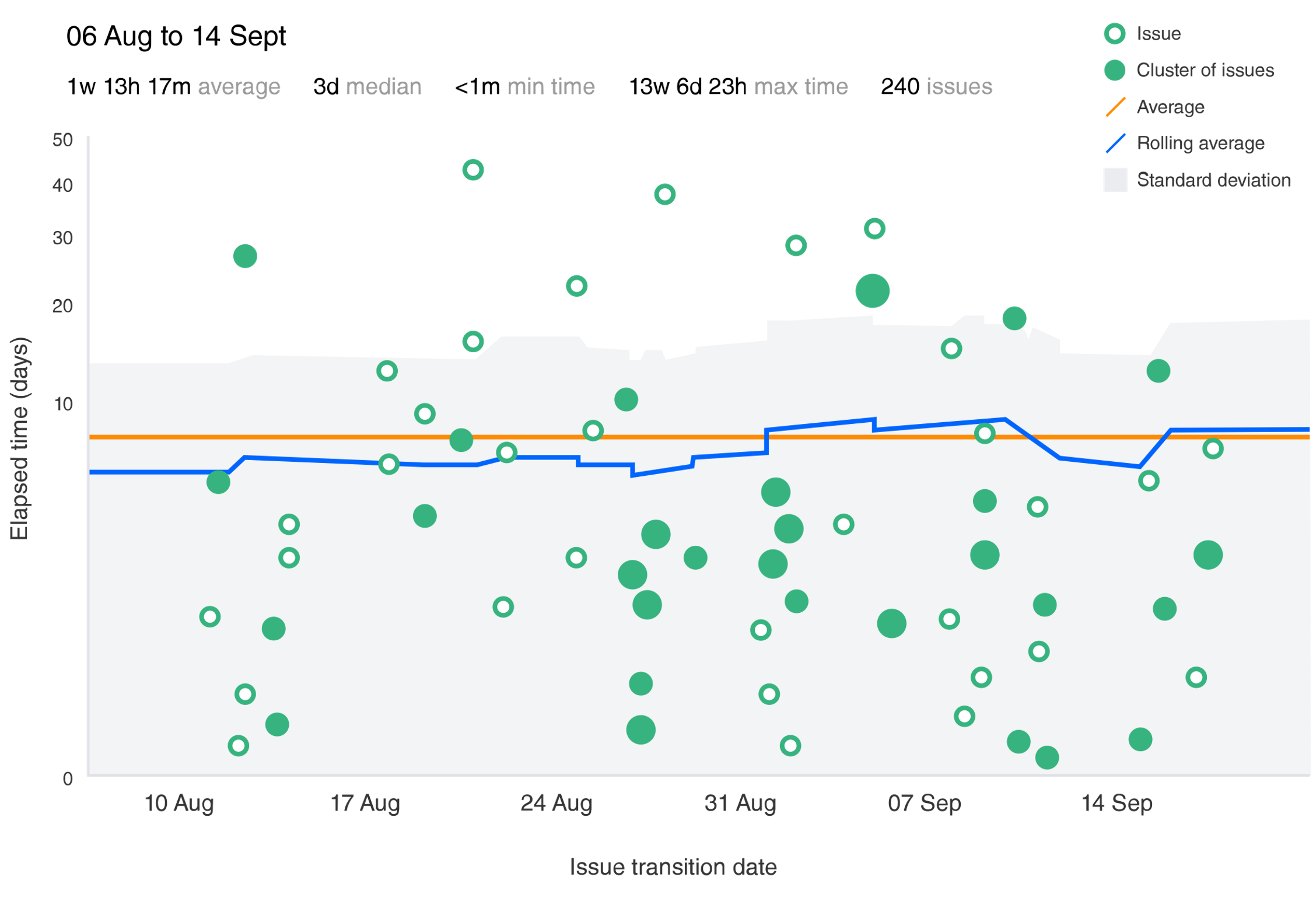
****

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

**Algorithmic Approaches**

* Programming Complexity (<https://en.wikipedia.org/wiki/Programming_complexity>)
  + McCabe’s cyclomatic complexity metric
  + Halsteads software science metric, etc.
* Software Quality (<https://en.wikipedia.org/wiki/Software_quality#Measurement>)
* Software Development Effort Estimation (<https://en.wikipedia.org/wiki/Software_development_effort_estimation>)

**Ethics**

* Hawthorne Effect - <https://stackify.com/track-software-metrics/>
  + Changes in behaviour due to knowing you’re being observed
  + “Metrics inform out incentives and shape behaviour” – Abi Noda

**Conclusion**

<https://blog.pragmaticengineer.com/can-you-measure-developer-productivity/>

**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].

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

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].

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. [↑](#footnote-ref-1)
2. If interested, the talk itself can be viewed at <https://www.youtube.com/watch?v=cRJZldsHS3c> [↑](#footnote-ref-2)
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> [↑](#footnote-ref-3)
4. See <https://waydev.co/> [↑](#footnote-ref-4)
5. See <https://linearb.io/> [↑](#footnote-ref-5)
6. See <https://jellyfish.co/> [↑](#footnote-ref-6)