**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**

* See KPI and Metrics email

The first section of this report will look to explore *how* you can broadly 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 drawbacks. Note that there are endless ways to measure software engineering and this section is no way exhaustive but rather scratches the surface in terms of the different metrics being used today.

**Source Lines of Code (SLOC)**

A basic software engineering metric used to measure an engineer’s output by counting the number of lines code in a source file. Although this seems like a rather simplistic metric, source lines of code (SLOC) introduces a feature common to many engineering metrics, a lack of clarity. For example, when we say lines of code, do we mean 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 of this measure 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.

**Pull Requests/Commits?**

Taken from GitHub Universe 2019

**Commits**

Small, frequency commits support greater transparency, collaboration, and continuous delivery.

Use cases:

* Reward teams with high number/frequency of commits
* Improve teams with low number/frequency of commits

Doesn’t tell you anything about value.

**Pull Request Count**

Poor measure of output:

* Doesn’t factor size or effort required of work
* Encourages unnecessarily small, chunked PRs

Good for understanding:

* Release cadence and continuous delivery

**Functions Point (FP)**

Whereas SLOC measures engineering activity from a source 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 necessarily 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, A., 2017).

**Code Churn**

As highlighted when looking at SLOC, quantity is not 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). The metric itself 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 skill or poor communication between engineers and the client which leads to ambiguous requirements.

For example, Engineer A may commit 100 lines of code and Engineer B may commit 30. By simply measuring SLOC, it seems like Engineer A has done more work. However, 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**

The next two metrics are specific to measuring engineering activity when Agile methods are being used. These metrics are used to help measure progress and to determine whether a team is working effectively.

**Team Velocity**

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**

This measurement 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**

**The elusive quest to measure developer productivity – GitHub Universe 2019**

<https://www.youtube.com/watch?v=cRJZldsHS3c>

Abi Noda – Senior Product Manager at GitHub – “flawed five” – all try to measure output and productivity and don’t work (16mins)

1. Commits
2. Lines of code
3. Pull requests
4. Velocity points – similar to team velocity covered above
5. “Impact” – abstract, basically lines of code with how many files were touched, how much is new code, etc.

* Measurements can be inaccurate because we collect and analyse what data is easily accessible – e.g lines of code, bug fixes
  + “Not everything that can be counted… counts” – Albert Einstein
  + Rather we should be analysing the metrics that are adding value to the product, not the metrics that are easy to get
  + e.g we could easily measure how many dishes a chef cooks but a better indicator of his performance would take into account how the food tastes, etc.
  + Graphical user interface, text, application, email

    Description automatically generated
  + <https://twitter.com/allenholub/status/1158524372111388672>
* Need to be able to define what constitutes as productive
  + Many people boil it down to output – but more is not always better – think about efficiency
  + People can game it and push quality down
  + If you think of engineering as an art, more paint is not better – not like a factory where you can easily measure how much you’re producing
    - But then how do you identify weakest links, etc.
* McConnell (says highest performing engineer is on average 20 times more productive than worst performing
* Gaming metrics – incentivise bad behaviours
  + e.g average code review turnaround time – encourages people to be less thorough to improve speed?
    - Found that people don’t put up code reviews on Fridays as the software used to measure productivity didn’t take into account weekends off
    - Quality of reviews fell
  + e.g pull requests – make smaller changes – therefore more pull requests for less work
    - Also could incentivise engineers to be less creative/take on larger problems
  + e.g velocity points – increase number of points task requires – makes estimates less accurate

**Good Measures**

Making metrics work – Noda

* Measure process – not output
  + Noda recommends teams measure processes over outputs to improve productivity
  + Code review turnaround time\*
  + Pull request size
  + Work-in-progress
  + Time-to-open
* Measure against targets versus absolutes
  + You define the targets – no two teams are the same, even within the same organisation – each team does their own time
* Avoid individual metrics – never seen that it’s beneficial enough to counteract the negatives it introduces
  + The presence of these measurements can influence teams negatively

**Development Analysis Platforms**

<https://www.sonarqube.org/>

<https://en.wikipedia.org/wiki/Software_development_process>

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

* Mentions Velocity and Pluralsight Flow

Jira – e.g automatically generate burndown chart

**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**

**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].

Noba, 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].

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].

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)