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

CSU33012 Software Engineering
Measuring Engineering

How Software Engineering is Measured and Assessed

Sean Gaffney, #19304695

November 26, 2021

1 Introduction

Software engineering in a complex and constantly evolving field which often makes it difficult to measure and quantify. This report attempts to examine the different steps of measuring engineering, including the ways in which software engineering can be measured, the data that can be collected and the platforms used to do so. It will also discuss the algorithmic approaches to process this data, and how different approaches can lead to different conclusions. It will discuss the ethical concerns of this level of data collection and its potential implications. Finally, it will try and conclude if data and metrics alone capture the whole picture.

2 Measurable Data

High quality data is an incredibly valuable resource when attempting to analyse software engineering activity. There are many approaches to measuring engineering activity, each with its own drawbacks and upsides. Each must balance simplicity and ease of collecting, with complexity and ethics. Not all approaches are equal, as often it is questionable whether the data collected is even relevant to answering questions about engineer output and progress on a project.

The most simple, and likely the most common measure of software engineering is commit and code change frequency. With version control systems such a git used ubiquitously in the industry, it is trivial to track and graph the commit frequency as well as lines added and removed by each developer on a project. This data is useful on the surface level to supervisors as well as programmers themselves, as it lets them track progress at a glance. However there are obvious issues with relying on these metrics to track progress. For example different engineers may have very different commit strategies, some may commit for every minor change made to a function, while others may only commit all changes at the end of the day. Regardless of which is the better strategy, both engineers could have the same output while the results would be challenging to compare.

Code churn has similar issues. Code churn is the number of lines of code that were added, deleted, or changed of a given period. For example one engineer could be working on refactoring existing code or bug fixing, and have only changed and removed dozens lines of code at the end of the day. While another engineer on a feature sprint may have written

hundreds of lines of new code. Again both may have put in an equal amount of effort but the results may look vastly different at a glance.

Testing coverage is another way to track what a developer has produced in a given period. Testing is a vital part of confirming the reliability of a piece of code, that the code will run consistently and without error in all situations. All modern production pipelines including testing, either written before or during development. This means that the number of tests passed are a simple way to track progress and gauge the overall progress and quality of a project.

Time and task tracking are more abstract but equally valuable ways of tracking progress. Time tracking is simply counting the number of days or hours spent on a project or a particular task. Task tracking refers to the process of breaking a large project into individual tasks that are more approachable. In agile development this often means using a Kanban system, where individual tasks are given to various developers, and tracked through multiple stages of development. There are a wide variety of systems available that allow task and time tracking to be easily integrated with the development environment including in JIRA and Github.

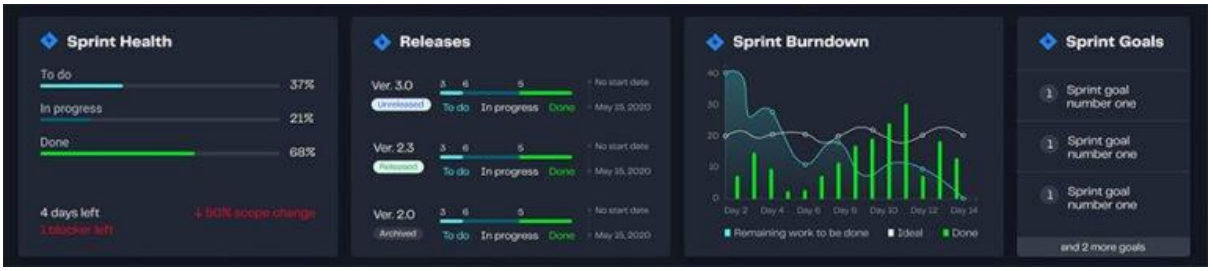
It's clear that there is a multitude of qualitative data that can be recorded in software development. However no single data type paints a clear picture of productivity and progress on a project. Combining data into a single source, so as to compare and contrast metrics, as well as context for the data is vital for drawing useful conclusions about the status of developers performance and where improvements can be made.

3 Computational Platforms and Infrastructure

There is clearly a wealth of readily available data that can be collected in software engineering. Consequently powerful tools are needed to process and analyse this data so that useful conclusions can be drawn. In some cases the platforms on which data is collected can also provide tools to collect and view this data, in other cases, third-party, or custom solutions can be used.

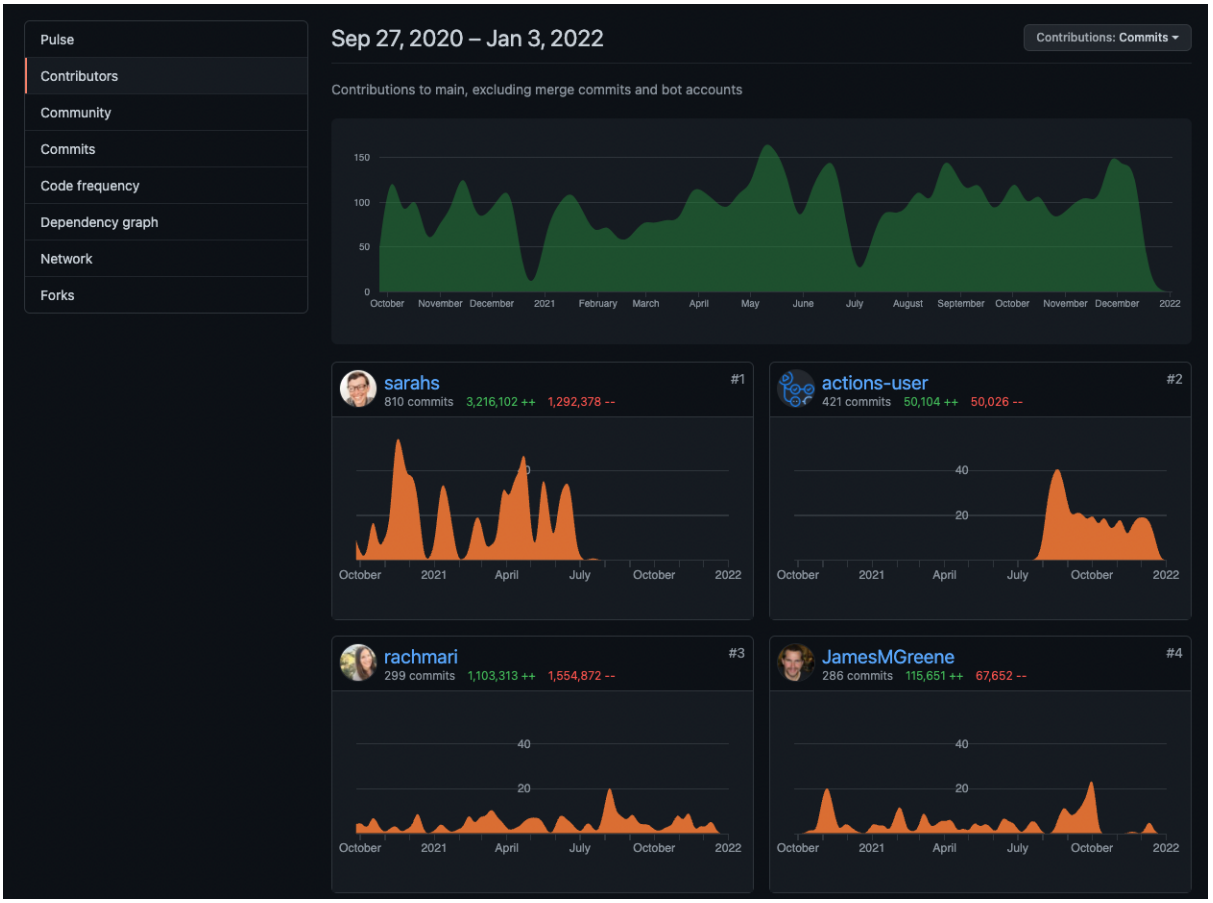
Atlassian is a leader in this space as it offers a number of project management tools that can aggregate data to share and display data for all members of a team. Bitbucket is their Git management tool, and Jira is their general project management tool that allows data about a project and its developers to be aggregated in one place. Jira can be used to generate

reports and view data in a multitude of ways to provide insight. Jira's platform can track metrics such as velocity, the total work remaining in a sprint, and version reports that display what tasks make it into each version.



A snapshot of a Jira dashboard for a team working in the Scrum framework

Github, the largest Git repository service, provides more basic data on their website, such as tracking commits and code churn. However Github's API is extensive and makes it easy for third parties to create their own platforms with the wealth of data available, alternatively some larger companies choose to create in house platforms that best fit their needs. GitHub hosts a marketplace where users can easily integrate trusted third party tools into their workflow. Popular measurement options include those supplied by Code Climate and ZenHub. These offerings when used in conjunction with GitHub's included analytics offer a compelling alternative to the Jira ecosystem.



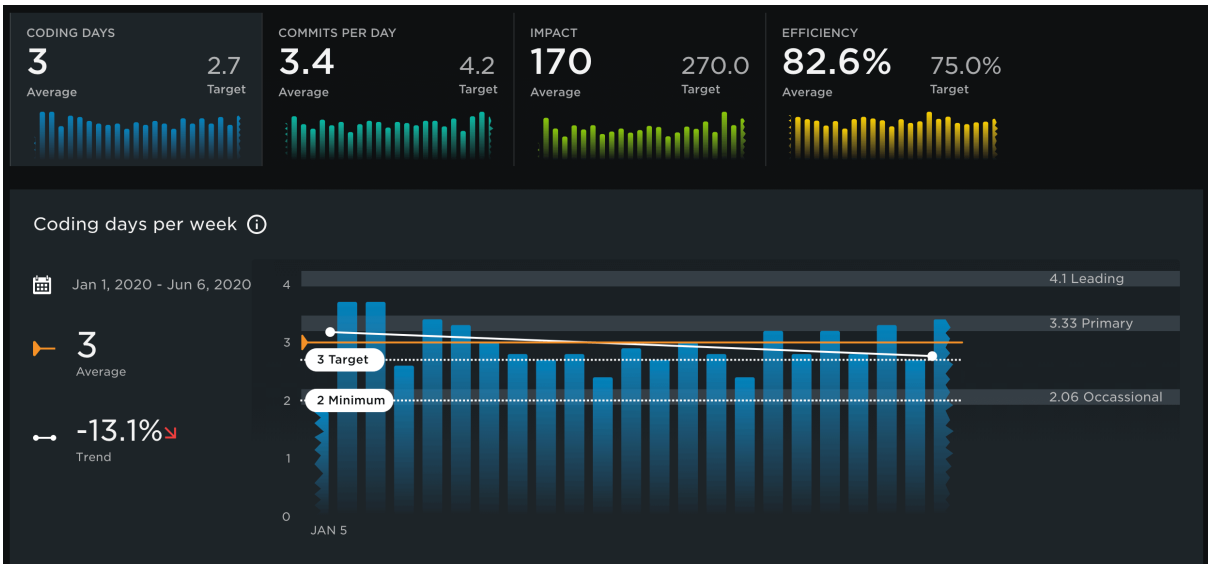
Github's built in tracking of commits and code churn by each contributor. This example is from GitHub/docs

Pluralsight is another third party collaboration and analytics platform. It can be integrated with a variety of Git management tools including Bitbucket and Github. Pluralsight's Flow application can not only display data such as commits, tasks, and time spent, but can further use this data to draw complex conclusions into how developers are using their time and which developers are most skilled in specific topics. It also derives metrics from the base data such as commit complexity, which attempts to measure the cognitive load associated with a given bug fix, and commit efficiency, which measures the percentage of commits that are new code and not just churn.

Teams and organizations have a variety of platforms to choose from when trying to better understand their data. They can go with a vertically integrated system such as Jira, use a third party tool in conjunction with a cloud Git management tool, or every create their own from scratch with the wealth of raw data through APIs. Any organization will need to find their own balance of the overhead involved with maintaining more complex tools, and the value they receive from relevant metrics and analytics.

4 Algorithmic Approaches

There are various techniques that can be used to profile the performance of a software engineer. As discussed above there are many platforms that can provide advanced metrics derived from basic Git and time data. Simple data, like time spent, commits, and code churn is used to create complex metrics, both to calculate individual and team performance. Organizations or managers will often set goals for engineers for a given time period ex; Ship feature X, or double the number of internal users of an API. With concrete goals in hand, managers can either find existing metrics through the platform their organization uses, find third party tools, or create their own using raw data and novel algorithms. For example Pluralsight Flow can display productivity over time and calculate if a project is on track or not. It can measure efficiency, project trends, and draw conclusions about both individual developers and entire teams and products.



An example of a weekly report generated for a single engineer with Pluralsight Flow. It includes simple data such as coding days and commits/day, but also derived

The next step in engineering analysis is the use of machine learning to draw more complex conclusions and even make predictions about performance. Modern machine learning models have the potential to find patterns in large amounts of data that traditional algorithms and metrics may not pick up on. There are already many other examples of this in other industries, from image recognition, to stock trading, fraud detection, language generation, and many more. Given enough data it's clear that machine learning could create new ways of discovering patterns in engineer productivity. For example AI may be able to make decisions on how to best divide limited resources, and where each engineer may be most

effective in a team or organization. However it’s not clear when tools leveraging machine learning will become widely available and exactly what effect it will have in this market.

5 Legal, Moral, and Ethical Concerns

It’s clear that there is a wealth of data available that can be applied to measure software engineering in great detail. On this surface this would appear to be completely beneficial, as being able to collect and process large amounts of data allows organizations to make informed decisions and increase productivity. However there are potential issues that can arise if the proper precautions are not taken.

The main moral issue with measuring software engineering is the invasion of privacy. To use the tools and algorithms discussed above to their full potential, personal data is collected about each engineer. These statistics must remain confidential and secure as allowing data to be seen. Commit and code churn data on an engineer are often publicly accessible within an organization should not need any special permissions. However tools that closely track activity and time spent may be more that engineers are willing to share besides with their direct superiors. In extreme cases engineers could find the level of tracking has gone too far. There is the potential that too much data collection may create unnecessary pressures to meet arbitrary goals and could actually lead to a decrease in productivity.

Legal issues regarding measuring software engineering stem from data security regulations. Due to GDPR and other other similar privacy regulations around the world, it's critical that data regarding engineering performance is protected. Systems to track engineers should ideally be run on well protected internal servers in an organization or a trusted cloud provider and use proper encryption and data security best practices. In any case, engineers should have full control over their own data and who it gets shared with. Ideally there should be a clear level of privileged access to make data leaks unlikely.

Even when data is properly handled and stored, engineer’s may question whether the data, and the metrics derived from it, are an accurate representation of their abilities and output. Organizations could have all the data in the world, but if they come to incomplete conclusions using said data, they may be harming engineers. It's not just a question of fairness, if management decisions are made based on poorly conceived metrics, engineers' pay, promotion, or even future job opportunities could be at stake. This is especially true in

the near future when organizations will likely base more decisions on machine learning models, which will likely be more opaque than traditional metrics.

6 Conclusion

There are many modern platforms and solutions for measuring software engineering. These tools allow engineers and organizations to track performance in a variety of interesting ways, and this can clearly lead to an increase in productivity and collaboration. However organizations must constantly re-evaluate what data they collect, if the conclusions they draw are accurate, and any potential downsides. No matter how much data is collected, it will always be hard to quantify performance without context, and making decisions solely based on metrics will never be beneficial. No matter how productive individual engineers are, there are always other factors in any organization that cannot easily be measured from team compatibility to organizational culture and structure.

7 References

Atlassian. 2021. *How to Use Jira Software | Official Buyer & User Guide*. [online] Available at: <https://www.atlassian.com/software/jira/guides> [Accessed 15 December 2021].

GitHub. 2022. *Contributors to github/docs*. [online] Available at: <https://github.com/github/docs/graphs/contributors> [Accessed 3 January 2022].

Hypercontext. 2021. *Engineering goals: How to set goals for high-performing teams | Hypercontext*. [online] Available at: <https://hypercontext.com/blog/work-goals/engineering-goals> [Accessed 21 December 2021].

Pluralsight.com. 2022. *Help your 1:1s with 3 Flow reports*. [online] Available at: <https://www.pluralsight.com/blog/teams/3-flow-reports-to-help-your-one-on-ones> [Accessed 1 January 2022].

Pluralsight.com. 2021. *Introduction to Code Churn: Causes & Fixes | Pluralsight*. [online] Available at: <https://www.pluralsight.com/blog/tutorials/code-churn#:~:text=Code%20Churn%20is%20when%20a,(typically%20within%20three%20weeks)> [Accessed 20 December 2021].

RoninPixels. 2021. *Guest Post: A Jira Dashboard for Samurais... or Ronins - RoninPixels*. [online] Available at: <https://roninpixels.com/jira/a-jira-dashboard-for-samurais-or-ronins/> [Accessed 20 December 2021].

TechBeacon. 2021. *Should you use AI to make decisions about your software team?*. [online] Available at: <https://techbeacon.com/app-dev-testing/should-you-use-ai-make-decisions-about-your-software-team> [Accessed 20 December 2021].

The Pragmatic Engineer. 2021. *Can You Really Measure Individual Developer Productivity? - Ask the EM*. [online] Available at: <https://blog.pragmaticengineer.com/can-you-measure-developer-productivity/> [Accessed 21 December 2021].