MEASURING SOFTWARE ENGINEERING - MICHAEL HASKINS, MSISS, 18323076

The aim of this report is to outline what measurable data is available and how it can be used to measure Software Engineering(SE), to provide an overview of what platforms are available to perform this work, to list the prominent algorithmic approaches available and the ethical concerns arising from this kind of analytics.

MEASURABLE DATA

SOURCE LINES OF CODE (SLOC)

Before researching this area, the first thought that popped into my head about measuring SE was to measure the lines of source code written per day, and to assess the consistency of this output over a period. However, the more I thought about this and after researching the topic I came to the conclusion that this is an incredibly crude and misleading way to measure SE. Counting the lines of code tells you nothing about the quality of the code, its difficulty, the amount of bugs in it, its importance, etc. It is missing the goal completely.

"Measuring programming progress by lines of code is like measuring aircraft building progress by weight" - Bill Gates

"At the time that people began using SLOC as a metric, the most commonly used languages, such as FORTRAN and assembly language, were line-oriented languages. These languages were developed at the time when punched cards were the main form of data entry for programming. One punched card usually represented one line of code." (Source lines of code, 2020). This was easy to count, and it made sense to use this as a measure for a SEs productivity, but programming languages have become far more complex and thus this measure has become extremely outdated. This measure is also very inappropriate to compare programmers who program in different languages, for example a program in Haskell is often much shorter than a program in Java which tackles the same problem.

This metric is outdated and should only be used cautiously, and any readers of the results from analysis of SLOC should be informed of its limitations.

CODE COVERAGE

If a program has high code coverage, this means that most of its source code has been executed during testing, which means the chance of the program failing or containing undetected bugs is far less than that of a program that has low code coverage.

This metric heavily relies on there being good testing and that consistently a large number of possibilities are being covered. If they are not, this can paint a false picture of the quality of the code being delivered by the SE, and there may be substantially more bugs than the code coverage suggests. This is a major shortcoming of the metric.

BUGS PER LINE OF CODE

Another metric for analyzing the quality of the code being written is discovered bugs per line of code, of course this is nearly always going to be an understatement of the real number of bugs per line of code, as many bugs go undetected, but it can be a useful metric for measuring the quality of testing and code being written by SEs. Better testing in particular would result in less bugs per line of code.

A drawback of this metric is that its hard to compare different programs as they will have varying degrees of difficulty. Rarely ever are two programs the exact same and each program presents different challenges that must be overcome. The more complicated the problem the more likely there is to be bugs in the code, as it is easier for the developer to overlook certain cases that the program will be exposed to. This limits the usefulness of this metric considerably.

BUG FIXES

Fixing bugs is an essential task for any software engineer. This can be nightmare for any software engineer if they are not familiar with the code or the code is not well documented, as to fix a bug you nearly always need to have a solid understanding of the program the bug originates from or in fixing it you can do more harm than good.

Fixing bugs as a result is often a tedious and slow process. Even if you have a solid understanding of the code and the problem, it can take some time to produce a patch as you have to be sure that your fix does not result in any more bugs. A good bug patch should result in no reopening (when a bug that was reported as fixed but has been reported again) and create no follow-on bugs.

A major drawback of this metric is how do you quantify the difficulty of a bug patch? Do you take into account whether the SE, or team, possesses any understanding of the base code?

GITHUB/BITBUCKET - VERSION CONTROL DATA

Version Control has become a staple of SE. It is used throughout the industry and has made globalization of teams and software companies possible. SEs typically use version control both for work and personal projects. This means there is a huge amount of data being stored and this can be used as a metric for SE.

There are many relevant and insightful metrics that can be drawn from version control data. Take GitHub for example, merely by interrogating the GitHub API it is possible to see the number of contributors, number of forks, number of stars, number of pull requests of a project/repo. You can get data on what; projects an individual is working on, skills an individual has and skills that are common among developers for a specific company.

Below is a visualization of the git commit history of the Linux Kernel Development on Friday, 18th May 2012. A video is available through the link here. Hopefully this gives an idea of just how much information can be drawn from version control platforms to measure SE.

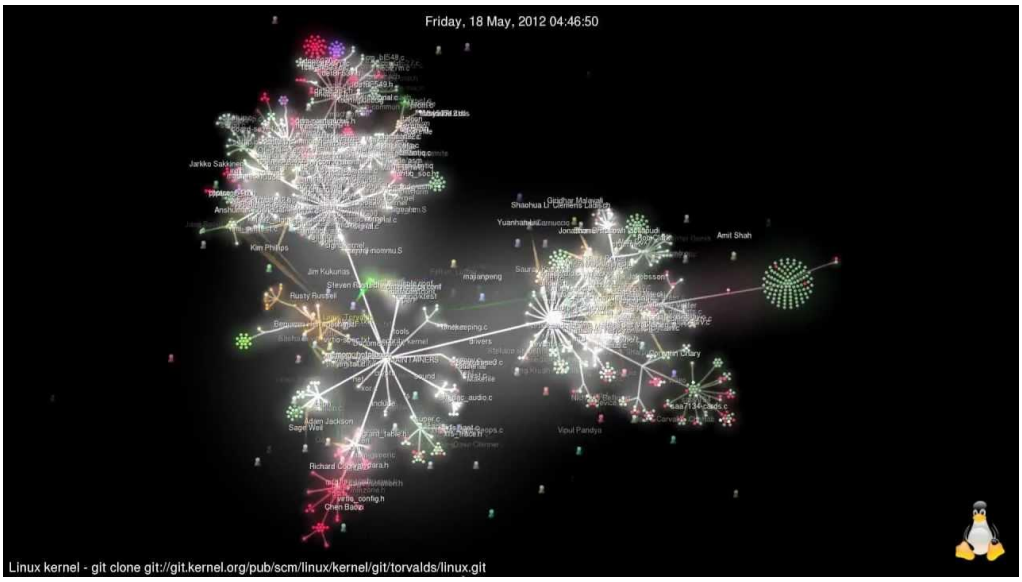


Figure 1 (Linux Kernel Development Visualization (git commit history - past 6 weeks - june 02 2012), 2012)

CYCLOMATIC COMPLEXITY

Is a metric for measuring software quality. It can be used as a standard to determine how many test cases are needed and to limit code complexity. (McCabe, 1976).

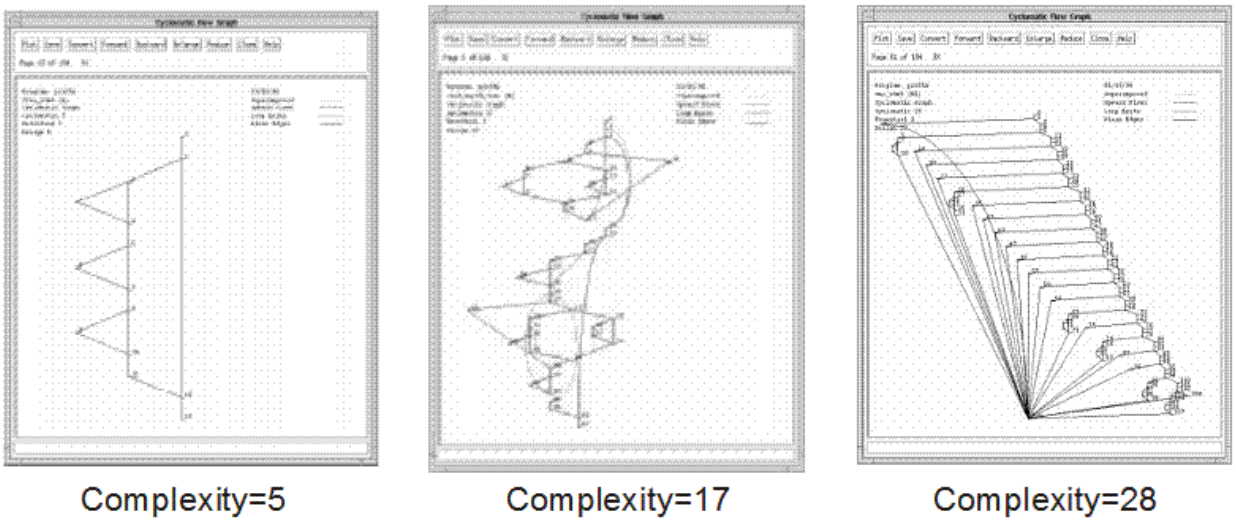


Figure 2 examples of program control flow graphs from NIST publication 500-235

The cyclomatic complexity of a program's source code is the number of linearly independent paths (each path has at least one edge that is not in one of the other paths) within it. The higher the cyclomatic complexity the more complex the code. Note in the above program control flow graphs the visible increase in number of possible paths (from top to bottom in each graph) as the cyclomatic complexity increases. McCabe recommended breaking the code down into smaller modules when the cyclomatic

complexity of that module exceeded 10. Complex code is harder to maintain and is viewed by many as being of low quality.

It is therefore prudent to ensure that the complexity of a module does not exceed a CC score of 10 and if it does management should work on simplifying the code as this will improve its reliability and make it easier to maintain.

EMPLOYEE HEALTH & ENVIRONMENT & HAPPINESS

It is important to remember that Software engineers are human and so, like anyone else, their productivity is affected by their health, happiness and working environment. These three factors can greatly affect the productivity of that individual. A healthy Software Engineer who is happy, has good communication with his colleagues and has a comfortable working environment will typically be more productive that a SE who is not.

Much data can be collected on an employee's health (with their consent). There are devices that can record the posture, heart rate, blood pressure, conversations and interactions of an individual.

Likewise, it is possible to collect much data which can be used to measure an employee's happiness. All the following can be recorded quite easily with modern technology: Conversations and interactions, the amount and quality of exercise the employee is engaging in and how much free time they have.

It has been found that total minutes of exercise per week was positively related to happiness (Zhang, Z., Chen) and also that happiness and particularly physical exercise is related to job performance (Drannan, a., 2016).

COMPUTATIONAL PLATFORMS AVAILABLE

CLOUD-BASED PLATFORMS

PLURALSIGHT (FORMERLY GITPRIME)

Pluralsight own the leading developer productivity platform. It analyses data collected from the version control system (E.g. GitHub, BitBucket, GitLab, etc) being used by the software engineer or software company. The amount of data that is available from version control systems is staggering and Pluralsight analyses this data in varying ways to provide useful insight into a software engineer's or team's productivity.

Its "primary features include workflow/performance review, collaboration, data aggregation, dashboard, historical data and project timeline. Its leader board functionality ranks engineers based on multiple metrics including daily commits, impact and efficiency. Additionally, its Work Log module collectively displays team performance, the contribution of members and work habits via interactive graphs." (GitPrime Software - 2020, 2020).

Below is an example of the way data drawn from version control systems can be visualized in an interesting manner by Pluralsight.

There are a number of similar platforms to Pluralsight such as Waydev, Code Time and Velocity (by Code Climate).

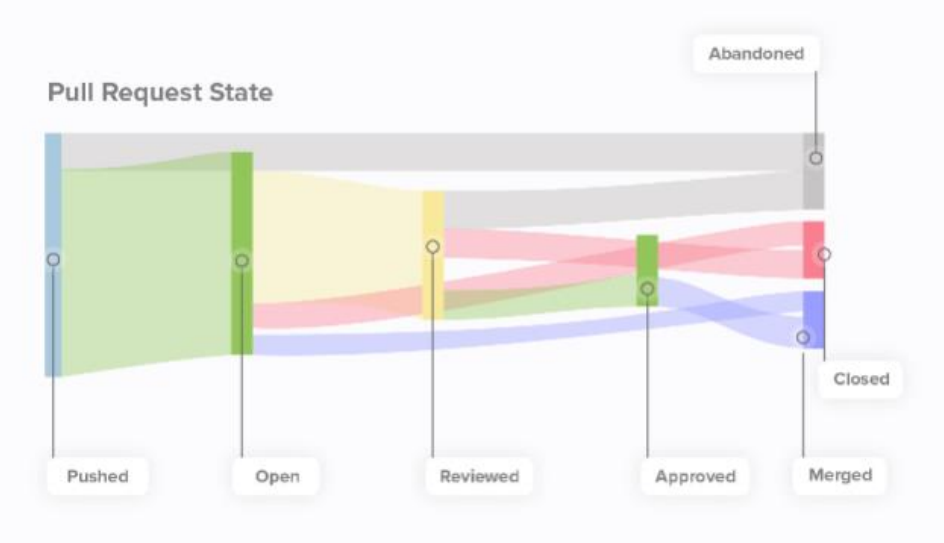
VELOCITY & QUALITY BY CODE CLIMATE

Code Climates engineering intelligence platform is called Velocity, they also have an automated code review platform called Quality.

Velocity is similar to Pluralsight in that they draw data from a client's repositories, but unlike Pluralsight whose main aim is to analyse the performance of individual software engineers, its main focus is to identify bottlenecks in the development process of teams or departments.

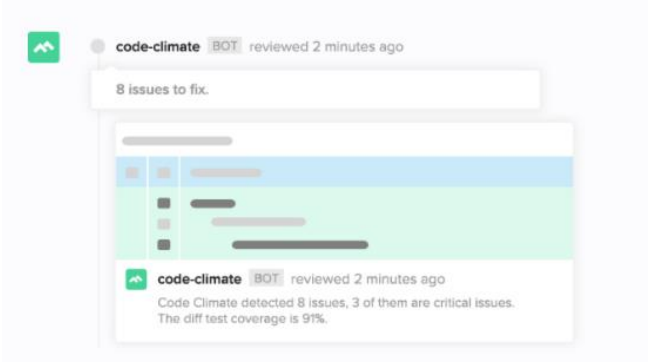
According to Code Climate's website Velocity provides features such as commit-to-deploy visibility. This enables management, no matter their level of their technological skills, to identify bottlenecks

throughout their development process. “Find your biggest slowdowns by pinpointing exactly where pull requests get stuck on the journey from open to deploy”. Below is a visualization of a pull request state which can be used to identify bottlenecks.



Velocity provides objective metrics which make it possible to see how changes affect the development process and make it possible to see if the team or department is trending in the right direction across a variety of metrics.

Quality is an automated code review platform. It provides data regarding test coverage, maintainability and more so that software engineers can save time and merge with confidence. It provides developers with real-time automated code review comments on their pull requests. Real time feedback enables software engineers to ship better code faster according to the Code Climate website. It also enables management to identify files with inadequate test coverage or maintainability issues.



FRAMEWORKS

HACKYSTAT

Hackystat is an open source framework that can be integrated by Software engineers, teams, or departments. It is used for the collection, analysis, visualization, interpretation, annotation, and dissemination of software development process and product data (Hackystat).

Hackystat collects data about development by means of 'software sensors' that software engineers can attach to their development tools. This data is sent to a web service called the Hackystat SensorBase where it is stored. Queries to the SensorBase repository can be sent by other web services and visualizations of the raw data can then be made.

ALGORITHMIC APPROACHES AVAILABLE

ARTIFICIAL INTELLIGENCE (AI)

The AI field is growing at a very fast rate. Machine learning (ML), which is a subset of AI, enables systems to automatically learn and adapt from experience without being explicitly programmed. Machine Learning algorithms can be divided into four groups.

SUPERVISED

A supervised ML algorithm is fed with training data containing the input/predictors and is given the correct answers. This enables the algorithm/computer to learn the patterns. It is then able to predict the output values for new data based on the patterns (relationships) which it should have learned from the previous data sets (Fumo, 2017).

Supervised ML algorithms can be used to classify datapoints into groups. In a Multivariate Analysis module I have taken in college I encountered the K-Nearest Neighbours supervised ML algorithm, which if it were given data describing individual software engineers performance (like commits, bugs per line, test coverage, etc) could be used to group the software engineers of company into a number of clusters and these clusters could then be analysed. Visualizations of these clusters could then be made and given to management with the hope that interesting insights could be drawn from them.

UNSUPERVISED

Here the algorithm is trained with unlabelled data. It is particularly useful when the human who runs the algorithm does not know what to look for in the data (Serokell, 2020). It can identify patterns that the human may have missed.

An unsupervised ML algorithm could be used to identify groups from the raw data that can be drawn from the GitHub API for example. This resulting groups can then be visualized and informative patterns in the data might be visible, like for example Software engineers who know more than say four programming languages might be involved in more projects, have more commits per day, less bugs per line, etc than software engineers who know less than four languages. Management could then begin

upskilling engineers who only know a handful of languages which would hopefully boost their productivity.

SEMI-SUPERVISED

This is merely a mixture of the two above groups, where the input data is a mixture of labeled and unlabeled samples. Once again, the algorithm can find patterns and make predictions itself.

REINFORCEMENT LEARNING

Algorithms of this kind use observations gathered from the interaction with the environment to take actions that would maximize the reward or minimize the risk (Fumo, 2017). These algorithms continuously learn from their environment in an iterative fashion until it explores the full range of possibilities.

Facebook has developed an open-source reinforcement learning platform called Horizon. The platform uses reinforcement learning to optimize large-scale production systems. Horizon has been used by Facebook internally:

- to personalize suggestions
- deliver more meaningful notifications to users
- optimize video streaming quality. (10 Real-Life Applications of Reinforcement Learning, 2020)

Reinforcement learning could also be used to offer useful code completion suggestions to a developer. This could be done by feeding the algorithm input from the developers or teams repositories/code-base which it could learn from and identify patterns which would enable it to predict what you are likely to type next after a certain line or what you likely what to do after you type the letter a for example. Microsoft's Visual Studio IntelliCode "uses machine learning to offer useful, contextually-rich code completion suggestions as you type, allowing you to learn APIs more quickly and code faster" (Caldwell, 2019). While not directly measuring software engineering this would likely boost productivity which is the aim of measuring software engineering, so that is why I have included this in this report.

BESPOKE APPROACHES

What the likes of Pluralsight do is to interrogate the version control repositories of their client, extract large amounts of raw data, and subsequentially to visualize data in a useful and meaningful way. What visualizations and insights Pluralsight generate may vary for each client and so the algorithm used will need to be customized to fit the needs of the client. This type of work could also be done easily enough by individual software engineers.

STATISTICAL APPROACHES

Data could be fed into statistical analysis tools like R, MatLab, SAS, etc and could then be analysed. R for example is good at generating engaging and highly customizable visualizations and these can deliver

useful insights to management. An R-markdown (RMD) report is an easy and professional looking way of creating a statistical analysis report.

ETHICAL CONCERNS

DATA COLLECTION

It is very important to carefully pick what metrics you include in measuring the performance of Software Engineers. Metrics must be chosen that cannot be easily gamed or manipulated by software engineers looking to inflate their performance score. For example, if you only measure software engineer's performance by how many source lines of code they write some will start writing long winded code that originally, they would have written in a far more condensed manner. If my manager started measuring my performance based off source lines of code written I would write programs I could have written in Haskell in Java, which would result in more long-winded code that essentially has the same function. This would result in an increase in my performance score, but I would be producing no more work than I was before, and more lines of code mean the program is harder to maintain and increase the likelihood of bugs.

It can be very challenging to get software engineers to buy into any plan for measuring software engineering. To ensure a good buy-in from the team the tools being used to capture the data must require minimal or no set-up, this is particularly why gathering data from cloud based version control systems is becoming the most common way of gathering data as it requires no set-up by individual team members. It is also important that data is being used to improve the overall software engineering process of the team or organization. This should be communicated to the team who may have concerns about its usage. Transparency is key.

Gathering the personal health data of employees is unethical unless employees opt-in. Management could then incentivise them to improve their physical health with the hope this would lead to an increase in their productivity at work. Rather than record employees blood pressure etc perhaps it would be better to have workplace initiatives which encourage and incentivise employees to improve their physical health. It would be extremely unethical to discriminate (e.g. leave them off the insurance plan) against employees if they have existing health conditions or of poor health.

DATA STORAGE

Any data, be it company or personal, will need to be stored securely. If there is a breach this may seriously harm employees trust in management and can damage the company image as a whole.

In 2018, the EU introduced the General Data Protection Regulation (GDPR). According to the GDPR website it "is the toughest privacy and security law in the world". This means that management can only store data as long as necessary, data must be stored securely, consent must be given, and a record of this consent must be stored along with the data. If GDPR is breached by management very harsh fines can be inflicted upon the company. It is therefore essential that management act within GDPR.

ANALYSING THE DATA

The idea of using AI to assess the performance of humans may not be a very welcomed idea. Do you trust the algorithm and the metrics used to make employment decisions? I wouldn't, this is because AI can be misled. Take the example of Microsoft's Tay. Tay was an AI twitter chat-bot that, according to Microsoft, learned from having conversations with users, the idea was the more conversations it had, the smarter it would become and learn to engage people through "playful conversation".

Unfortunately, Tay was corrupted by users who fed it, to say the least, misleading information and within a day it became a Nazi, misogynist, racist and Donald Trumpist. It was quickly shut down and became probably the most well-known case of an AI program failing (theverge, 2016).

This highlights the risk of using AI. AI could easily be misled by software engineers if the metrics it was using could be easily gamed or manipulated.

MISINTERPRETATION OF THE DATA

The insights provided by the analysis should not be used as justification for employment decisions, as a misinterpretation of the data could wrongfully cost someone their job. A good developer might write very condense code and so not score very well if source lines of code was used as a metric, even though they may be contributing just as much or more than other members of the team who score highly in this metric. It is therefore important that the people interpreting the insights generated from the data understand the context of it and can determine that the data is not misleading.

The resulting data from the measurement of Software engineers or teams can be used for a variety of things such as determining the quality of the current product or process, improving that quality and predicting the quality once the software development project is complete (What are Software Metrics?

Examples & Best Practices, 2017). It can also be used to "understand which teams and individuals are the strongest at a given task or type of project, you can better understand not only how to assign projects, but where your organization has weaknesses and where your team could benefit from more training as well" (Wallgren, 2020).

BIBLIOGRAPHY

- En.wikipedia.org. 2020. *Source Lines Of Code*. [online] Available at: <https://en.wikipedia.org/wiki/Source_lines_of_code> [Accessed 24 November 2020].
- Zhang, Z., Chen, W. A Systematic Review of the Relationship Between Physical Activity and Happiness. *J Happiness Stud* **20,** 1305–1322 (2019). https://doi.org/10.1007/s10902-018-9976-0
- Drannan, a., 2016. The Relationship between Physical Exercise and Job Performance: The Mediating Effects of Subjective Health and Good Mood.
- 2012. *Linux Kernel Development Visualization (Git Commit History - Past 6 Weeks - June 02 2012)*. [image] Available at: <https://www.youtube.com/watch?v=P_02QGsHzEQ> [Accessed 24 November 2020].
- McCabe, T., 1976. *A Complexity Measure*.
- StackShare. 2020. *Github Vs Gitprime | What Are The Differences?*. [online] Available at: <https://stackshare.io/stackups/github-vs-gitprime> [Accessed 24 November 2020].
- Softwareadvice.com. 2020. *Gitprime Software - 2020 Reviews, Pricing & Demo*. [online] Available at: <https://www.softwareadvice.com/ie/app-development/gitprime-profile/> [Accessed 24 November 2020].
- https://hackystat.github.io/#:~:text=Hackystat%20is%20an%20open%20source,Researchers.
- Fumo, D. 2017. *Types Of Machine Learning Algorithms You Should Know*. [online] Available at: <https://towardsdatascience.com/types-of-machine-learning-algorithms-you-should-know-953a08248861> [Accessed 25 November 2020].
- Serokell. 2020. *Artificial Intelligence Vs. Machine Learning Vs. Deep Learning: What'S The Difference*. [online] Available at: <https://medium.com/ai-in-plain-english/artificial-intelligence-vs-machine-learning-vs-deep-learning-whats-the-difference-dccce18efe7f#:~:text=Machine%20learning%20is%20a%20subset,that%20help%20to%20solve%20problems.> [Accessed 25 November 2020].
- *Neptune.ai*, 2020. 10 Real-Life Applications of Reinforcement Learning.
- Caldwell, C., 2019. *AI-Assisted Intellisense For Your Team'S Codebase*. [online] Microsoft. Available at: <https://devblogs.microsoft.com/visualstudio/ai-assisted-intellisense-for-your-teams-codebase/> [Accessed 25 November 2020].
- https://gdpr.eu/what-is-gdpr/
- The Verge. 2016. *Twitter Taught Microsoft'S Friendly AI Chatbot To Be A Racist Asshole In Less Than A Day*. [online] Available at: <https://www.theverge.com/2016/3/24/11297050/tay-microsoft-chatbot-racist> [Accessed 26 November 2020].

- Stackify. 2017. *What Are Software Metrics? Examples & Best Practices*. [online] Available at: <https://stackify.com/track-software-metrics/> [Accessed 26 November 2020].
- Wallgren, A., 2020. *Should You Use AI To Make Decisions About Your Software Team?* | *Techbeacon*. [online] TechBeacon. Available at: <https://techbeacon.com/devops/should-you-use-ai-make-decisions-about-your-software-team> [Accessed 26 November 2020].