Measuring Software Engineering Report

CSU33012

John Sinclair – 16325734

Table of Contents

Introduction

Measuring software engineering is an incredibly important part of the software engineering process. Software engineering is defined as “the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software” by the IEEE (Institute of Electrical and Electronic Engineers). Without measurement, projects can easily go over budget, require more time than expected and deliver less functionality than originally planned. With proper measurement systems in place, it allows a management team to accurately plan a project in advance, giving accurate estimation of cost, time, and functionality.

The measurement process itself is a valuable tool in the hands of a competent management team. Understanding how and when you or team works best is essential to productivity. Problem is how can we accurately assess such a complex and non-linear task? There is no standardised metric applied sector-wide and the methods used have seen little to no improvement through the development of the profession. Simple methods such as measuring a programmer’s lines of code or number of commits do not accurately portray the effectiveness of a programmer. So other, inventive methods have been devised.

Top tech companies use an array of tools and metrics to help them quantify what makes a good developer, giving their managers the ability to give feedback on a regular basis and often developers will have a scheduled performance reviews once or twice a year to update the developer on their performance, especially in relation to the company’s standards and expectations.

In this report I will be delving into the various and unique methods companies use today, its effectiveness, as well as the tools and platforms at their disposal. I will also be giving an overview of some algorithmic approaches to measuring software and the ethics inherent to these analytics.

Measurable Data

Lines of Code

While lines of code are very easy to measure, it has little to no meaning when evaluating a software engineers’ productivity or effectiveness. As Bill Gates said, “Measuring software productivity by lines of code is like measuring progress on an airplane by how much it weighs". Lines of code only measures the volume of non-whitespace lines added to a repository or codebase. This means that it can only be used to compare projects that are using the same frameworks and same coding standards. When either of these options change the length of the codebase changes and so cannot be used for comparison.

Another issue with the lines of code approach is its inherent rewarding of adding to the codebase's volume. Edsger Dijkstra believed that software engineering should be approached by determining how many lines of code were "spent", and that a software engineers' role should be to subtract, not add lines of code. When coders are efficient, they write far less lines of code but under this form of measurement they would be looked upon as worse than their inefficient counterpart, pumping out lines of inefficient and verbose code.

A semi-useful twist on lines of code is code churn. Code churn is when an engineer rewrites their own code in a short period of time. As we know the raw data of how many lines of code written is not a useful metric, but code churn slightly improves upon it. Code churn measures the total net addition to a project's codebase, i.e., the total amount of productive code added by an engineer. So, if a developer wrote 10 lines one day, then rewrote 8 of them the next, their lines of code would be much greater than their productive lines of code added (18 vs 10). This provides a more accurate picture of a developer efficiency but still suffers from the main pitfall of lines of code, rewarding code inefficiency.

Number of commits

Similar to lines of code, the number of commits can be easily measured thanks to the services provided by version control systems like GitHub and Bitbucket. It has a similar measurable weight to lines of code as the quality of the work done has no relevance to the number of commits made or the impact it had on the project.

Another reason why the number of commits isn't an accurate way to measure a developer's work is that most software engineers have differing habits when it comes to committing to a project repository. A developer who commits after every small feature addition of bug fix will be improperly favoured over another who commits large codebase updates at the end of long programming sessions.

A better data point to measure the importance of a developer's addition to a project is the commits content and by extension, their overall commit history. While more resource intensive a manager will get a far more accurate picture of the benefit the engineer has provided to the project, however, we still encounter the issue of quantifying this contribution.

Errors and bugs

Human nature dictates that errors are an inevitable part of the software engineering process, but the number of bugs introduced by a developer cannot alone be used to judge their quality. Bug density is a far better way of evaluating a developer or engineer. Bug density refers to the average amount of bugs found in a developer's code, often in a

standardised measurement like 100 or 1000 lines of code. So, if developer A introduced only 1 bug whereas developer B introduced 10, the number of bugs would determine that developer B is the less effective engineer but say developer A only wrote 10 lines of code and B wrote 1,000, B is by far the superior developer with just 1 bug per 1000 lines of code, vs. A's 100.
This method of evaluation is still flawed as not all bugs are created equally. A few small user experience bugs may be far more preferable to one fatal bug that takes down an entire service. This means our bug density metric is not necessarily a good measurement of an engineer's effectiveness, however it can be indicative of their quality, but again this will be up the discretion of the manager.
While number of bugs introduced can be indicative of the quality of an engineer, it is certainly not a great indicator of their effectiveness. Even if a developer introduces a number of bugs, if they are also closing a number of bugs in the same timeframe they are, in my opinion, effective.

Testing

Testing is an essential part of the software engineering process and is also an effective way to evaluate a projects development. When a codebase is thoroughly tested it is far more robust and less likely to fail making it a popular tool. The important metric when testing a codebase or a commit is code coverage. If there exist chunks of un-executed and therefore un-tested code the possibility of errors and bugs dramatically increases. It is common in companies for a certain threshold of code coverage to be reached before a pull request or merge can be granted.

It is important to note that while it is a very effective tool for evaluating a code base, the tests themselves must be comprehensive. Edge cases for unanticipated inputs must be accounted for and only then is it an accurate measure of robustness.

Employee health

Another important data point to measure is employee health. The software engineering process entirely relies on the engineers themselves and so if one or many are not operating at full capacity neither is the overall system. Independent analysis of the factors that contribute to reduced employee effectiveness such as sleep, and nutrition can prove valuable.

An obvious element of the software engineering job is to sit in front of a computer for long periods of time where the developer is exposed to large amounts of blue light. When exposed to blue light the brain delays the release of melatonin, a vital chemical that triggers the onset of sleep. Another chemical all developers are familiar with is caffeine. The highly common and effective stimulant found in coffee and most energy drinks is very good at keeping you awake. Caffeine blocks the production of adenosine, a protein that builds up throughout the day. When adenosine builds up unabated throughout the day the high levels of it accumulated by the evening causes drowsiness. The problem with caffeine intake is the length of its effects. Its half-life is said to be between 3 and 5 hours meaning you may still feel its effects come bedtime.

This all matters because a lack of sleep has been shown to lead to cognitive impairments so much so that the Centre for Disease Control and Prevention (CDC) in America has declared insufficient sleep a 'public health problem'. If lack of sleep notably decreases cognitive function it most definitely affects the quality and efficiency of software development.

Employee health metrics clearly can be used to measure the software engineering process, but they do give rise to ethical concerns over how intrusive the monitoring of employees can be.

Computational Platforms

Source control tools

At the most basic level, source control tools such as git provide users and teams with insights into many different facets of their projects. Data points such as contributors' lines of code added and deleted, commits and code frequency are all readily available within the insights section of GitHub but without third party software this information can be near impossible to interpret accurately.

Flow

Pluralsight's product Flow aggregates historical data from your projects git repository and provides useful insights and reports to help make your team work more efficiently. Pluralsight claims that when using their Flow product, projects will see an increase in coding days per week, an increase in commits per day as well as increased impact to the codebase.

One particularly interesting feature in the flow suite is the team effectiveness insights provided. These insights are language specific with info such as commit efficiency and what they call "Skill IQ" data which determines the efficiency and quality (think skill level) of the code committed. Flows main aim is to make it easy for management to access valuable facts and metrics on their developers' performance.

Jira

Jira is an issue and project tracking software. It is designed with the agile software development process in mind, allowing for easy visualisation of all yet to be started, in progress and completed sprints. It integrates with Atlassian's git-based source control product Bitbucket and so make the Jira with bitbucket couple a commonly a commonly used tool set by agile developer teams and provides functionality to connect the codebase with the project's overall roadmap. This can be useful to track a projects growth over time and can provide increased estimation accuracy of future feature additions or projects. Jira also allows for prioritisation of certain sprint elements and tickets, optimising the workflow of the team with added transparency. While Jira provides insight into a developer completed tasks, it still requires entry of reports and completed tasks, adding a manual element to the software development process that reduces efficiency.

Waydev

Similar to Flow, waydev offers is a toolset to analyse a projects codebase and the efficiency of its developers. For management, its advanced and comprehensive tools streamline the evaluation process of developers. They split their offering into three main categories.

The first is a day-to-day visualisation of every team members contributions and work habits. This daily stand-up feature makes it easy to focus in on a developer and see all their commits and pull request made so they can make better decision, set achievable expectations. The second is their improved one-to-one meetings functionality. Having accurate metrics and data for every developer means that one-to-one meetings can accurately reflect and address a developer's performance. The third is its code review workflow, where the code review process is given a sense of clarity. They provide a bird's eye view of pull request activity allowing management to gain a complete picture of the process and optimise developer collaboration.

Waydev improves upon more manual tools such as Jira by automatically generating reports for managers use. It provides a developer summary for managerial use which has numerous valuable metrics:

- Impact – a way to measure the weight and importance of the code changes happening, much more advanced than methods like simple line of code analysis.
- Throughput and productive throughput – referring to the total amount of code, be it new, churned or refactored. Productive refers to the proportion of code committed without churn.
- Efficiency – what percentage of a developer's code is productive. The higher the efficiency the longer the code has been providing business value. Important to note that a high churn rate will decrease this metric.
- Technical debt – the amount of code refactoring done by the person.

Hackystat

Hackystat is an opensource framework that is used for the collection, analysis, visualization, interpretation, annotation and dissemination of software development process and product data. Its users typically attach so called sensors to the tools they already use for development for the collection of data that is then sent to the Hackystat SensorBase for storage. This database can also be queried by other services to generate interpretations of this raw data. It can be integrated with source control tools like git so that users can gain visual insights into their codebase. Some concerns arise from the use of Hackystat and its automated data recording, employees can find it unsettling when faced with uncertainty over how much of their data is being collected.

PROM

PROM is a similar service to Hackystat that provides data analysis for developers but is more comprehensive, catering to not only the individual user but also whatever form of management team is in place. PROM only shares gathered data and interpretations with the developer themselves and their management team. This eliminates one of the largest concerns over using a service such as Hackystat, the data itself is transparent, readily available to the developer to understand how their data is being collected and interpreted.

Algorithmic Approaches

COCOMO Model

The constructive cost model (COCOMO) is a regression model based on lines of code. It was proposed by Barry Boehm in 1970 and was based of 63 projects. It splits a project into two core parameters, effort and schedule, that will determine the overall quality of the project. Effort refers to the amount of labour required to complete the task, measured in person-month units and schedule refers to the amount of time needed to complete the task.

There are different forms of the COCOMO model to account for the variance in different software projects. The first form, the basic level of the model can be used for quick and rough calculations of software costs. the accuracy of this form is its neglecting of certain important factors, a trade-off of its quick deployment. The second form 'intermediate' takes into account these extra drivers of cost. The third form 'detailed' additionally accounts for the influence of individual phases in the project.

Boehm also outlined a framework for determining what type of system a project was, split into three categories: organic, semi-detached and embedded.
An organic system is a software project that has a team that is adequately small, a well understood problem that has been solved before and a team of experienced developers in the problem area. An organic system is the least complex of the three systems.
An embedded system is the most complex of the three systems, demanding high levels of complexity, creativity and experience. An embedded system requires the largest team size of all three models.
The middle ground system, a semi-detached system, is a system that lies between the organic and embedded system models. It is said to be a semi-detached system if the

components such as team size, complexity and creativity requirements are more than that of an organic model but not as advanced as an embedded.

Due to the three system classifications as well as the three different models, there is a COCOMO model to suit the needs of any software development project.

Deep Code

Increasingly implementation of machine learning and artificial intelligence into the code review process is becoming very common. A company called snyk provides a product called deep code which they claim is the first real-time semantic code analysis tool. Having used it myself I was very impressed with the clarity of the suggestions. It draws upon all its users to amalgamate a general knowledge base that trains the system to recognise common coding practices, errors and shows suggested fixes, powered by its knowledge base. It offers tools that integrate both with your IDE and your source control system, allowing for automatic code review of pull requests or before code is committed. While this is not a direct measurement of software engineering it is perhaps an applicable metric that is at the disposal of the management team. The ability to significantly increase code review times and accuracy with very simple integration should be of significant use to a development team.

Ethical Concerns

Many ethical concerns arise from measuring the software development process. Personal privacy in our society is at an all-time low and is only trending lower. Threats to our personal privacy are inbound from a variety of fronts ranging from governments and their agencies spying on their own citizens, to large multinational companies forensically monitoring their employees’ day to day lives with increasing intrusiveness.

The most unethical form of measurement is via secret or subversive monitoring of employees without their knowledge or consent. In my opinion this is unacceptable and all employees that are being monitored by any means should be informed on how so. However, using services such as waydev or flow are in my opinion perfectly fine, in fact, encouraged. I believe it is entirely within the right of a company to ensure no team or team member is under performing and so the metrics that these services provide, evaluation of the quality and efficiency of a developer’s code is entirely ethical.

In August of 2018 the company Three Square Market made headlines when it microchipped many of its employees. The chips are used for a range of functions like opening doors, unlocking computers and paying for items in a vending machine. The company have also come out with an app that tracks the location of the microchip, putting it to use on released prisoners out on probation, doing away with ankle monitors in favour of these grain of rice sized chips. It’s worth noting the CEO says the company won’t be using the technology to track employee whereabouts stating that “there’s no reason to”. It does however give rise to questions about other companies and CEO’s who have different opinions.

The tech industry has been a clear antagonist in general personal data privacy with companies like Facebook gathering large masses of data on its users to more efficiently sell them products via advertising. Amazon is another whose direct monitoring of its employees has gained public criticism. Amazon has recently been granted patents to track an employee’s location within a warehouse and to monitor hand movements. According to Amazon warehouse employees they were constantly updated on their performance, and how they needed to increase performance (by gathering items quicker). James Bloodworth, the author of “Hired: Six Months Undercover in Low Wage Britain”, said that during his time at an amazon warehouse he felt that in order to meet the company’s performance target he felt the need to run between items, an act which resulted in a disciplinary from management. In my opinion this cannot be good for a human mind, this form of zero-sum game constantly throughout the day would result in a high level of stress for the employee.

Stress has been shown to have many negative effects on humans such as heart disease and high blood pressure. On the other hand, I believe it fair that a company is within their rights to demand a certain standard of their employees, in fact an employee should expect to be held to a specific standard but in my opinion these performance targets and standards should not be created and met at the expense of the employees’ health and mental well-being. The particularly aggressive methods by amazon have not to my knowledge made it to the software engineering profession as of yet but it begs the question when they will and in what form will they arrive.

There are many other companies such as Teramind and VeriClock that provide services to track employees’ emails, keystrokes, web usage and even take pictures via their webcams. The Times reported on Chinese companies that are using helmets to monitor workers brainwaves in order to detect fatigue and emotions and in one case of an electrical company, the helmets would determine the duration of an employee’s break based upon a perceived amount of work done. This level of monitoring is undoubtably highly intrusive. While it may have positive effects like increased safety it also has negative effects like the sharing of private personal information, that is being entrusted to your management.

While all of this is legal, is it ethical? In my opinion it all comes down to communication. Business have the right to get the full potential out of the employees they pay however, it should all be done in an entirely transparent manner. An employee should be informed on all the methods used by the employer and how the gathered data will be interpreted and managed as well as the employee having full access to their data. In my opinion software engineering measurement systems should be used to add more value to the product for the client or end user, not to squeeze every dollar out of the human capital at the expense of the employees' health and well-being.

Conclusion

In conclusion despite the various methods of measuring the software engineering process there exists no one silver bullet to accurately evaluate the software engineering process. Each method or approach has both its positives and negatives, be they technical restriction or ethical boundaries. Currently the best approach is to use a number of different methods in tandem to give an accurate overall view of the process and when used effectively the act of measuring software engineering can be of incredible use to a project. Thanks to the wide variety and different tiers of intensity, there exists an option for every software engineering team.

There still exists the question as to how far employers can insert themselves into their employee's personal data. I am firmly of the opinion that employers are within their rights to monitor the business metrics exclusively, providing that they are entirely transparent with the owner of the data, on its contents and how it will be used.

References

MEASURABLE DATA

- https://blog.pragmaticengineer.com/can-you-measure-developer-productivity/
- https://wiki.c2.com/?LinesOfCode
- https://wiki.c2.com/?SubtractLinesOfCode
- https://www.pluralsight.com/blog/teams/why-code-churn-matters#:~:text=Code%20churn%20is%20when%20an,it%20again%2C%20and%20then%20again.
- https://www.mattlayman.com/blog/2019/how-sleep-affects-your-code/
- https://www.rand.org/pubs/research_reports/RR1791.html
- https://www.infopulse.com/blog/top-10-software-development-metrics-to-measure-productivity/

PLATFORMS

- https://www.atlassian.com/software/jira
- https://www.pluralsight.com/product/flow
- https://code.google.com/archive/p/hackystat/
- https://waydev.co/waydev-metrics/

ALGORITHMS

- https://semmle.com/assets/papers/measuring-software-development.pdf
- https://www.deepcode.ai/

ETHICS

- https://www.theguardian.com/world/2018/may/14/is-your-boss-secretly-or-not-so-secretly-watching-you