MEASURING SOFTWARE ENGINEERING

Cs3012 – Software Engineering

Maria Cairns 19334725

CONTENTS

INTRODUCTION

What is software engineering? And why do we need to measure it, and how?

This report aims to answer these questions.

As the name suggests, software engineering consists of two components; software and engineering. The former is defined by the oxford dictionary as being" the programs used by a computer for doing particular jobs", and the latter as "the activity of applying scientific knowledge to the design, building and control of machines, roads, bridges, electrical equipment etc". It is therefore fitting that the Institute of Electrical and Electronics Engineers (IEE) defines software engineering as "The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software". This definition is not far from the original definitions coined by Fritz Baur and Margaret Hamilton.

In the last 50 years, and even prior to the term being defined, there has been exponential growth in the field of software. Particularly as we move towards a more connected world, and the Internet of things, more and more of our everyday habits are affected by software. Another consideration would be the number of people entering the field from different backgrounds; those who grew up coding since they were a teenager, those who have a formal education in the subject, or those who may have pivoted in their career path and have gained accreditations through other means. All of these factors bring about the necessity for adequate quality measurement standards and testing.

However, there is not a one-size fits all approach when it comes to measuring software engineering. Due to the complexity of the subject, there are a number of different means of measurement. This leads us to the first section of the report; *How can one measure software engineering activity?*

HOW CAN ONE MEASURE SOFTWARE ENGINEERING ACTIVITY?

I believe that the means of measuring software engineering activity, productivity and quality are split into two categories: quantitative and qualitative. On the quantitative side, there are the data sets; the key performance indicators, the stats on how may lines of code were written, or how many commits were made to the repository etc. In other words, the things one can put a number on.

The qualitative element is the area which is more difficult to measure; for example how effectively the software engineer can work with their team? Do they slow progress by not communicating? Do they boost morale of the team leading to greater productivity and employee satisfaction?

I believe that due to the complexity of the field it is necessary for evaluations from both humans and algorithms. However, in practice, it is easier said than done. The scale of companies, the skillset of the staff evaluating the engineers, the faults in the measurement system or the gaming of these systems are just a few of the problems that are faced with this task of measuring software engineering.

So, moving on, what data can we measure?

SOFTWARE ENGINEERING PROCESS

The software development life cycle illustrates the different stages of development that software engineers have to go through when developing software. There are some variations in terms of the structure - for example waterfall or agile development, but the main elements remain the same.

Image source: synotive.com

The software development life cycle describes the stages of the development process for software. Whether in an agile, waterfall, or another structure the phases remain the same.

1. Requirement gathering and analysis: The requirements and the analysis of the focus of the project are conducted in this phase. Once the breadth of the requirements is defined, a requirements specification will be produced, allowing for the next phase of the model to be reached.
2. Design - using the requirements documentation from the first phase, the hardware and system requirements are drawn up to aid in defining the overall system architecture. The system design specifications are then used in the next phase of the model.
3. Implementation: This is where the coding begins. The work is divided into units, and this tends to be the longest phase of the cycle, depending on the project.
4. Testing and integration: Once a codebase has been developed it must be tested to check that it has reached the minimum requirements outlined earlier

in the cycle. Once this testing is successful the product is deployed to the customer for use. At the initial handover more testing will take place, and once the bugs are fixed this phase concludes.

5. Maintenance: Once the system is in use by the customer there may be some problems which arise. Dealing with these issues is the main component of the maintenance phase of the cycle.

This is a lengthy process, with each phase being the basis for the next. The length of this process and its different elements lends itself to the reasons behind why there is no one metric for how software engineering should be measured.

It should also be noted that things do not always go to plan, there may be unforeseen bugs, the project may not satisfy the client's requirements, or the project itself may be far larger than had been previously scoped in the requirements document.

MEASURABLE DATA

Moving onto the data that can be computed, often it is the computing and gathering that is the simplest part of the process. Interpretation and avoiding gaming of the system are where things get more complicated.

There are many factors that come into consideration when assessing the quality of a software engineer. Code quality is an obvious place to start, however there is many ways to measure this, and multiple debates over which is the most effective. Measuring the quantity-focused metrics also have their place, as do production metrics, measuring the amount of work done and the efficiency of the software engineer.

Agile process metrics can also be measured, which although don't quantify the software itself can be used to refine the software development process. These metrics concentrate on how agile teams plan and execute their decision making within the team.

CODE QUALITY

Lines of code

Lines of code (LOC) is one of the simplest and easiest metrics to derive when measuring software development. It was the original measurement, and is derived from counting the lines of code. One major fault is that it does not incorporate the actual contents of the code, and therefore gives potential rewards for lines of "dead" code and does not incentivise the optimisations of programs. An improvement of this metric is measuring source lines of code (SLOC). SLOC divides code into two

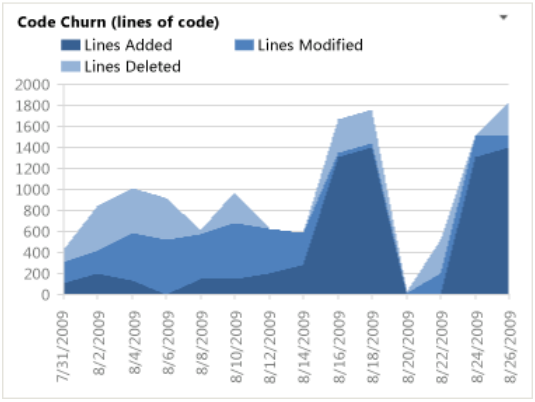
categories: physical and logical. Physical SLOC is a count of lines in the program's source code (including comments etc), whereas logical SLOC is a measurement of the number of statements. Although it does not solve all of the issues that the LOC measurement has, it is a useful measurement in relation to estimating the complexity of the software (i.e., more SLOC, more complexity). Another alternative or a modification of analysing LOC on their own would be to measure the proportion of the final project that each engineer writes.

LEAD TIME

Originating from the automotive industry, "lead time" is defined as the time it takes between a customer placing an order and receiving said order. A more appropriate definition in relation to this subject would be the time elapsed between the identification of a requirement and the and it's fulfilment. Lead time can be used to measure how responsive the customer the software engineers are, however a fault of this metric would be that it is omitting an imperative factor - the actual size of the project.

CODE CHURNING

Code churn is the number of lines of code that were modified, added or deleted in a specific period of time. Code churn can be useful in determining how much useful code an engineer writes. If there is a significant proportion of code churn for one engineer of the team, it may show a lack of experience or perhaps if the product manager did not define the task effectively etc. Looking at code churn alongside lines of code is a useful was to determine the amount of value that each engineer is bringing to the final result. Code churn can be tracked on GitPrime.



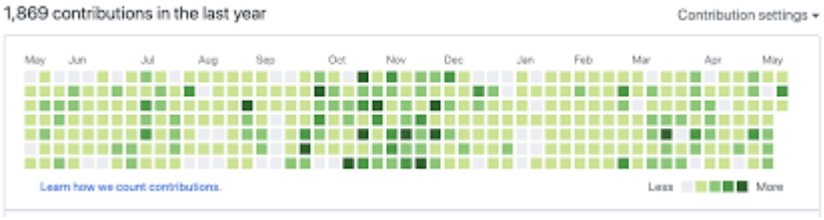
Code churning graph

VERSION CONTROL SYSTEMS

Version control systems (VCS) are a very important tool for storing files, recording changes to files and for multiple developers to collaborate on a project. There are many available for example GitHub, GitLab, and bitbucket. GitHub is the world's largest open source, social coding site.

There are many data sets which can be measured from a single VCS profile. The number of commits, languages used, how many followers one has, how many repositories they have committed to are just a few. Git commits in particular are a good indicator of an engineer's overall productivity. The graphical representation on GitHub is very clear but is not a true representation of an engineer's productivity. Similar to the issue with the LOC measurement, the actual quality of the code is not

represented. An engineer may be very active and commit code every day, but this code may have bugs which are never fixed, or take days for a team member to fix, reducing the productivity of the team instead.



The graphical representation of commits on GitHub

COMPUTATIONAL PLATFORMS

Moving forward from the data, and onto the platforms which can be used to share this data. There are a number of different platforms which allow software engineers to share their work in turn enabling the access and assessment of their code.

HACKYSTAT

Hackystat is an open-source framework for collection, analysis, visualization, interpretation, annotation, and dissemination of software development process and product data. It is a fully automated computational platform. Sensors are attached to the development tools to quietly gather data on the product development process and analyse it on a remote server. It can measure software engineers in real-time, which poses the ethical question of *what level of monitoring is too much? How much do companies really need to know?* These queries will be covered in the ethics section of this report.

PSP

Watts Humphery created the Personal Software Process, a structure that provides engineers with a disciplined personal framework for working on software. It provides a process in which engineers can plan, measure and manage their work. The size, time and defect data of a software project are collected and analysed. Although this is a beneficial process. For the modern developer it requires many forms to be filled out, and checklists regarding the tasks completed and any errors that may have occurred. The time-consuming process and the nature of human error make it less appropriate for use in a commercial environment. As the PSP process enforces reflection, and the analysis of one's own work it is a useful tool in a teaching or learning setting, but not far beyond it.

PSP consists of three stages, divided into many levels: planning, development (design, code, compile, test) and a 'post mortem' or review. In the final stage the developer records all of the data for the completed project. Based on this data, they

create their coding standard and a plan for further improvement, a “personal improvement plan”. This plan includes ways to improve the software development process further for that individual software engineer. As this is a highly personal metric its effectiveness would depend on the individual and their attitude towards the process. As a metric that could be scaled up across a company, it would prove difficult to assess due to its individualistic nature.

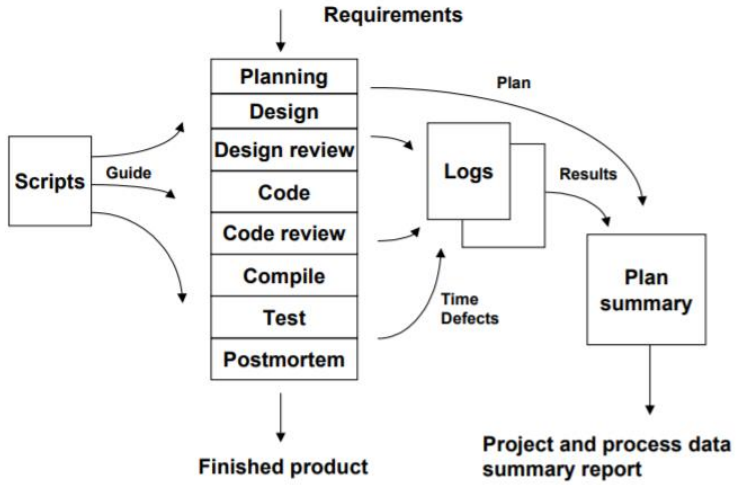


Figure 1: PSP Process Flow

PSP process flow (Humphrey, 2000)

PLURALSIGHT FLOW

Previously known as GitPrime, provides “deep insights” into the engineering workflow. It communicates the increased visibility of workflow patterns as a selling point.

It focuses on metrics closely related to lines of code, for example impact scores, which measure the speed and difficulty of line changes. It also comes with a fully featured dashboard with an array of different metrics, again posing concern in the region of how much monitoring is too much? Do team managers need access to all of this data? Competitors to Pluralsight flow such as Haystack.io are using their selective and less populated dashboard, with “as-needed” access to the different data points, as a selling point.

Pluralsight Flow allows engineering leaders to monitor engineers individually, with a focus on individual reporting and team ranking. This may not be a beneficial metric and may do more harm than good. According to a study conducted by Google, on team performance in *Project Aristotle* with a total of 180 teams, the study concluded that individual performance of team members is not significantly connected with team effectiveness, but psychological safety is essential.

WAYDEV

Waydev is another platform for tracking engineering team performance. Similar to Pluralsight flow it aggregates and displays critical project and developer metrics. It is an agile, data-driven method, which began in 2017 with the idea of adding analytics to software development.

SonarQube is code quality assurance tool that collects and analyses source code and provides code quality reports for the project. This tool can be used to ensure code reliability and the security of applications. Using tools like SonarQube can ensure that the code base is clean and maintainable. It can enforce many different coding standards, from testing, to performance, to complexity to documentation. I believe that tools like SonarQube provide valuable insights and ensure a standard of quality to the product but are most beneficial to those trying to measure software engineers when used in conjunction with other tools.

ALGORITHMIC APPROACHES

There are multiple algorithmic approaches to analyse the data collected during the software engineering process.

As mentioned in the beginning of this report, the complexity of software engineering and development makes it difficult to measure. Fortunately, the source of this problem also provides the solution. The branch of artificial intelligence, machine learning is one of the most powerful tools when it comes to analysing this data. Machine learning is a method of data analysis that automates analytical model building. It is based on the idea that systems can learn from data, identify patterns and make predictions with very little human intervention.

MACHINE LEARNING

SUPERVISED LEARNING

In supervised learning, input data is provided to the model along with the output. The goal of supervised learning is to train the model so that it can predict the output when it is given new data. Supervised learning is mainly used in classification and regression problems. These supervised learning models are less suited for handling complex tasks, and there are a number of other disadvantages which make it sometimes more difficult to implement, as it requires lots of computation times in its training, and we need enough knowledge about the classes of the objects.

UNSUPERVISED LEARNING

With Unsupervised learning the input data is provided, but unlike supervised learning, the output data is not provided. It has the aim of identifying patterns inferred from the unlabelled input data. It can be used for clustering and association problems. In the case of measuring software engineering, unsupervised learning may be more effective as it works on unlabelled and uncategorised data, and can find useful insights from the datasets. As we often don't have a conclusive answer to what metrics are indicative of a "good" software engineer based on the data collected, unsupervised learning models are useful for identifying correlations between the metrics which humans may not have encountered prior to this.

REINFORCEMENT LEARNING

The third branch of machine learning is known as reinforcement learning. It is based on rewarding desired behaviours and/or punishing desired ones. The reinforcement agent perceives and interprets its environment, therefore learning from the rewards and punishments. It acts as a way of directing unsupervised learning through rewards and penalties. There are many uses for reinforcement learning for example in robotics, gaming and autonomous vehicles. An advantage of this type of machine learning is that it can gather experience from thousands of parallel gameplays if run on software which has sufficient computing power. However, it can have some difficulties when in a real-world environment as there many changes in the environmental factors so it can make it harder to determine whether the outcomes were as a result of the changes in the agent, or the change in the environment.

GAMING OF THE SYSTEM

By measuring software engineering, one is essentially measuring the product of human behaviour. It is imperative that when calculating this measurement, one is aware of human nature and its natural tendencies. The Hawthorne effect is when people alter their behaviour when they are aware that they are part of an experiment. It can skew the results of an experiment and can be difficult to quantify or recognise at first. For example, a hand-washing study carried out on medical staff found that there was a 55% greater level of compliance when the participants of the study were being watched. In the context of software engineering, when engineers know they are being monitored on a particular metric, it is natural to assume that this may invoke a change in their behaviour. However, the benefit of this behavioural change can vary depending on the metric. If measuring lines of code for example, this may lead to less motivation for developers to write optimised code, as they are not being incentivized to.

ETHICAL CONCERNS

In 2018, the European Union (EU) developed the General Data Protection Regulation (GDPR), which aims to protect the personal data of EU citizens. It means that the personal data of EU citizens should only be stored where there is a lawful

basis, such as a legal obligation or the consent of the citizen. It acted as a substantial update of the Data protection act of 1988 and had a shift in focus to purpose limitation and data minimization. Personally, I am concerned by the lack of formal data protection legislation across the globe. For example, in the US there is no equivalent to the GDPR regulations, with only three states, California, Colorado and Virginia enacting comprehensive consumer data protection laws. It is a concern, that the largest economy in the world, and the home of so many of the technical institutions which have an oligopolistic control over how we live today does not have comprehensive data protection laws, and if one of the most developed countries does not have this regulation, how is the data of the citizens in the developing world being regulated?

With the rate of development of technology, the lethargic pace of lawmaking in this area is concerning. Without formal obligations for companies to comply with, they are left to their own devices in terms of acceptable data protection practices. In today's world, so many of these corporations do not act in favour of their workers or users of the product. The focus is on profit maximization, market share, whereas complying with data protection (if in place) is more of a tick the corporate social responsibility box situation. It is saddening that often scandals and breaching of these laws are the basis for change, as these are things that affect share price and revenues.

To quote Andrew Lewis "If you're not paying for it, you're not the customer, you're the product being sold". This infers that the value gained from companies having you use their product is greater than the benefit you gain from using the product.

I think that it is important to bear this in mind, as many of the tools used for measuring software engineering are not produced in house, with the rise of aforementioned platforms such as Waydev and Pluralsight flow, although most charge a fee, sometimes engineers may not be aware of the real cost, and where all of this data is actually going - not everyone reads the fine-print.

In relation to the metrics these platforms compute, I believe that there are both advantages and disadvantages. It is obvious that measurements of code quality, and overall productivity are beneficial to have when analysing the performance of engineers, but the level of detail required in these computations is where there needs to be some consideration for data protection. Measurements of how long an engineer sits at the desk, their bathroom breaks, the tone of their messages to co-workers all cross the border of the ethical considerations in my opinion. As humans, we do not act like machines, therefore using metrics which measure us like them is unethical. How can a machine learning model take into account how one engineer may boost another's performance? Or how would this model make an allowance for health issues faced by an employee? I believe that there must be a distinction between an employee's personal and professional in the workplace.

In my opinion measurements of productivity, from more than one metric (in an attempt to gain a fairer depiction) are useful and necessary when assessing a software engineer, for example monitoring the number of bugs fixed, the quality of their code, meeting attendance etc. However, more personal data points such as

bathroom breaks, personal notes, amount of time at their desk are more personal than professional therefore should be omitted from the evaluation.

I also think that despite the Hawthorne effect employees have the right to know exactly how they are monitored by the company, and how this data is processed and stored.

CONCLUSIONS

This report by no means provides an exhaustive list in terms of how one can measure software engineering, it is merely an exploration of some of the options available, the positives, and negatives, and the ethical considerations.

In the words of Peter Drucker, *"you cannot improve what you don't measure"* and it is clear that this rings true in the field of software engineering. There are many quantitative and qualitative measures of employee performance, and I am of the opinion that the optimal solution is a combination of the two. While there are many benefits to the computational data sets on employee performance, one should have a level of human oversight to the process too, to encompass the elements that these models cannot display.

Every engineer is a human, with multiple factors affecting their work. Therefore, the broader the model, the more accurate a depiction it will give of the engineer. By having multiple metrics measured by both humans and machines, a fairer view will be given as bias can be minimised. As long as the boundaries of privacy are not crossed when collecting data of software engineers, the measurement of software engineers is a logical and necessary procedure for corporations. Going back to Drucker's quote, the measurement of software engineering allows for improvements to the individuals, and is beneficial to the organization, while done in an ethical manner.

REFERENCES

https://www.castsoftware.com/glossary/what-is-software-engineering-definition-types-of-basics-introduction#:~:text=The%20IEEE%20fully%20defines%20software%20engineering%20as%3A&text=The%20application%20of%20a%20systematic,application%20of%20engineering%20to%20software.

https://www.oxfordlearnersdictionaries.com/definition/english/software#:~:text=%5Buncountable%5D,computer%20for%20doing%20particular%20jobs

https://waydev.co/software-development-metrics/

https://en.wikipedia.org/wiki/Gaming_the_system

https://www.macmillandictionary.com/dictionary/british/play-the-system

https://www.synotive.com/blog/software-development-client-questionnaire

https://medium.com/@jilvanpinheiro/software-development-life-cycle-sdlc-phases-40d46afbe384

https://pvs-studio.com/en/blog/terms/0086/

https://www.agilealliance.org/glossary/lead-time/#q=~(infinite~false~filters~(postType~(~'page~'post~'aa_book~'aa_event_session~'aa_experience_report~'aa_glossary~'aa_research_paper~'aa_video)~tags~(~'lead*20time))~searchTerm~'~sort~false~sortDirection~'asc~page~1)

https://stackify.com/track-software-metrics/

https://www.google.com/imgres?imgurl=https%3A%2F%2Fflaviocopes.com%2Fgit-squash%2FScreen%2520Shot%25202020-05-12%2520at%252016.26.46.png&imgrefurl=https%3A%2F%2Fflaviocopes.com%2Fgit-squash%2F&tbnid=_QNytOlNlzjwwM&vet=12ahUKEwiXy4f4ppT1AhXMF8AKHQRyDl8QMygLegUIARDHAQ..i&docid=ENR4Fg1TLWVJ9M&w=1505&h=435&itg=1&q=git%20commits&ved=2ahUKEwiXy4f4ppT1AhXMF8AKHQRyDl8QMygLegUIARDHAQhttps://www.pluralsight.com/blog/teams/why-code-churn-matters

https://hackystat.github.io/

https://ai.ia.agh.edu.pl/wiki/_media/pl:miw:2009:2005_johnson_-_improving_software_development_management.pdf

https://resources.sei.cmu.edu/asset_files/TechnicalReport/2000_005_001_13751.pdf

https://www.cs.umd.edu/users/aporter/Docs/challenges.pdf

https://www.usehaystack.io/haystack-alternatives/haystack-vs-pluralsight-flow-gitprime-a-pluralsight-flow-gitprime-

alternative?utm_campaign=Competitor%20Ads%202.0&utm_source=google&utm_medium=cpc&utm_content=Pluralsight%20Flow&utm_term=gitprime&gclid=Cj0KCQiAt8WOBhDbARIsANQLp950UVXmBWr9yoQwpt6l3Mcxvhu2mjqFJk_aH-ungDoLzvJSA9IDWDYaApryEALw_wcB

https://www.pluralsight.com/product/flow
https://waydev.co/about-us/

https://www.sas.com/en_ie/insights/analytics/machine-learning.html#:~:text=Machine%20learning%20is%20a%20method,decisions%20with%20minimal%20human%20intervention.

https://www.javatpoint.com/difference-between-supervised-and-unsupervised-learning

https://www.crestdatasys.com/blogs/an-introduction-on-using-sonarqube/

https://www.synopsys.com/ai/what-is-reinforcement-learning.html
https://neptune.ai/blog/reinforcement-learning-applications
https://www.techtarget.com/searchenterpriseai/definition/reinforcement-learning
https://martinfowler.com/bliki/CannotMeasureProductivity.html
Sackett Catalogue of Bias Collaboration, Spencer EA, Mahtani K, **Hawthorne effect**. In: Catalogue Of Bias 2017: https://catalogofbias.org/biases/hawthorne-effect/

https://www.citizensinformation.ie/en/government_in_ireland/data_protection/overview_of_general_data_protection_regulation.html#:~:text=In%20Ireland%2C%20these%20laws%20include,consent%20or%20a%20legal%20obligation.
https://worldpopulationreview.com/countries/countries-by-gdp

https://www.google.com/imgres?imgurl=https%3A%2F%2Fflaviocopes.com%2Fgit-squash%2FScreen%2520Shot%25202020-05-12%2520at%252016.26.46.png&imgrefurl=https%3A%2F%2Fflaviocopes.com%2Fgit-squash%2F&tbnid=_QNytOlNlzjwwM&vet=12ahUKEwiXy4f4ppT1AhXMF8AKHQRyDl8QMygLegUIARDHAQ..i&docid=ENR4Fg1TLWVJ9M&w=1505&h=435&itg=1&q=git%20commits&ved=2ahUKEwiXy4f4ppT1AhXMF8AKHQRyDl8QMygLegUIARDHAQ