Measuring Software Engineering

CS3012 - Software Engineering
Ruth Brennan | 17329846

Introduction

As business leaders continue to seek a new edge, many are turning to big data. Vast scores of data are being collected, analyzed, and applied to various processes throughout the organization to improve products and operational efficiency. They look to improving their developmental process. In the movie Moneyball[1], where the general manager of the Oakland A's, one day has an epiphany: Baseball's conventional wisdom is all wrong. He entrusts the future of his team in statistics and as a result the previously bottom of the table team rocket to the top. An example of how we can improve when we bet on statistics.

Statistician David Spiegelhalter poses the basic question [2] “How many trees are there on the planet?”. To answer this question, we first have to agree on what a tree is. Some would argue that a tree a bush or shrub should be included in the definition and others would state that there is a stark difference between a tree and a sapling. While discussing the variants and later going in to describe the final decision, Spiegelhalter made the excellent point that if authorities differ on what we should call a tree, it should be no surprise, that more nebulous concepts are even more challenging to pin down. For example, what constitutes a good approach to software development? What makes a good software engineer more productive than their peers? And the focus of this essay how should we go about measuring software engineering?

Why do we want to measure software development?

We first look at the why on a level of personal improvement and then we will proceed to look at why through the eyes of a company.
Allow me to draw a rather sensational comparison, Harold Shipman was Britain’s most prolific serial killer. Though he does not fix the usual criteria. He was a mild tempered, family doctor who worked in the suburbs of Manchester. He injected over 200 of his elderly patients with a huge opiate overdose. He was only caught when he made the mistake of forging a will and

[1] Moneyball, directed by Bennett Miller based on a book by Michael Lewis
[2] Spiegelhalter’s 2010 Pelican Books publication, The Art of Statistics

got caught. In a subsequent inquiry into whether he could have been caught sooner it was revealed that his "excess deaths", compared to the GPs in the surrounding locality of Shipman totalled at almost exactly the number of people later confirmed to be victims. That is not to say that this would always be the case but it does show the importance of us allowing data to mast red flags. We would enable ourselves to stop issues before they become project serial killers by simply analysing our data.

At the end of the day there is an undeniable truth to a sentiment perpetuated by Carl Jung "Knowledge rests upon truth alone, but upon error also"[3]. We have all undoubtably at some point been told to learn from our mistakes, after all it is the most tried and true way to improve. So if available data can point out mistakes we don't know or have yet to realise we are making, the real question is why would we not let it?

We then turn to look at this from the perspective of a company. It is fair to say that the aim of most software development companies is to make a profit. According to Business gateway, There are four key areas that can help drive profitability.

These are **reducing costs**, **increasing turnover**, **increasing productivity** and **increasing efficiency**. To demonstrate how measuring and analysing software engineering allows us to achieve those goals I will employ a series of examples.

If we take a look at testing. Traditional testing relies heavily on the technical and business requirements of the project as a baseline to assess against the code. Is the product meeting the specifications outlined prior to the project beginning? This approach is very inward focused as opposed to user focused. It's a "let's produce a good product, launch it to the market, and hope our assumptions were correct" approach. Now, with tons of data at our fingertips, traditional testing methods have become outdated and are being replaced by a more proactive approach to testing. For example, companies are instituting feedback loops to collect user usage data. They're also collecting enormous streams of data from every step of the testing process for analysis. Arguably one of the most impactful changes to testing is the use of predictive analytics to anticipate user's needs and make adjustments to products based on those predictions. This vastly improves a user's response to a product in a positive way.

Another positive way for a company to improve it's productivity is to look at it's employees. It would be naïve to say that every employee has the same

[3] Quote from 'The Creativity Code' by Marcus Du Sautoy. Published by 4th Estate London

skills, ability and drive to complete their job to the highest level. I imagine the skills, ability, drive and overall productivity of employees could all be depicted as bell curves. The majority of workers find themselves in the mean section of that curve, the engineers willing to go the extra mile on a project, work overtime or those who work hard to refine their skills find themselves at the top and those who fail to do so find themselves at the bottom.

The saying that a chain is only as strong as it's weakest link rains true when we are talking about employees in a company and indeed members on a software development team. We cannot ignore external factors that may be contributing to a lack of productivity at the moment of analysis. Perhaps the employee has fallen ill, maybe they have come to dislike their job or they could have a complicated home life. Maybe they are not capable of preforming the level required of them in a particular instance. Looking at the big picture, it is important that a company is able to identify it's weaknesses in order to improve the chain as a whole. In an instance where an employee is underperforming for whatever reason it becomes an issue for the overall productivity of the company but data enables companies to identify these instances and to combat them. For example if a team of engineers were about to embark on a project developed mainly in python and data identified a particular engineer whose python skills were substandard to the rest of their team then it would be in the company's best interest to promote improving their skills before embarking on this project, offer training to this employee and give them the opportunity to upskill or they could swap that engineer out for one with the required skills. If a situation were to arise where a company had to let some employees go it would only stand to reason that they look at the data and shed those at the bottom of the bell curve in favour of retaining those that are more productive.

The police use the term CompStat to define using data to identify problem areas in a neighbourhood. They then attempt to rectify these problems with a targeted approach. That is essentially the summative why we would want to measure software development. To identify and target problem areas. Now that we have established why it is we want to collect and use this data we need to consider where we can find it.

Where can we find the data?

We can start by looking at data that is readily available to us:

1. STATE OF AN END PRODUCT

We can learn a lot about the engineers and engineering methods employed in developing a piece of software from the finished product. We can look at the surface: if the clients are happy that their requirements have been met and that they received a solid return on their investment then that would suggest that the project was a success. We can gather further information by looking at different aspects of the final design.

For example if we can look at the **percentage of code programmed by an individual that remains in the project**. This attempts to measure the amount of productive code contributed by a software developer. The amount of churn shows the lack of productive code. Thus a software developer with a low churn could have highly efficient code.

We can also tackle measuring software yield by counting the number of text lines in a programmers code (**LOC**). While the slightly better alternative is to look at source code and work off the Source Lines of Code (SLOC), of which there are two main types; physical and logical. Where physical is a count of lines in the program’s source code. It includes lines with comments and blank spaces. Logical SLOC is a count of the number of statements.

At the point of completion we are able to determine the **Lead Time**. This quantifies how long it takes for ideas to be developed and delivered as software. Lowering lead time is a way to improve how responsive software developers are to customers.

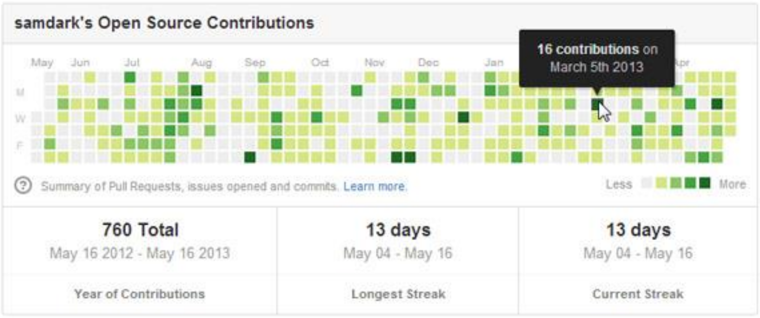
Cyclomatic complexity is a software metric used to indicate the complexity of a program. It is a quantitative measure of the number of linearly independent paths through a program's source code which is supplied to us via the end product.

2. DATA FROM VERSION CONTROL SYSTEMS (VCS)

Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later. Remote repository services such as GitHub, GitLab and Bitbucket can be used as platforms to gather useful metadata on software projects, in addition to their roles as spaces for collaboration and backing up of code. GitHub is the world's biggest social coding site. It's a place where developers can host, share and collaborate on code that is built around an open source framework.

We can use VCS to **identify problems**. When combined with other continuous integration tools we can tell whether or not a committed piece of work builds or not. So if a programmer regularly commits code that doesn’t work and fixes it afterwards or leaves other programmers to fix it then we can use this method to identify them.

The **number of commits** is indicative of how active a programmer is in a project. It would allow us to clearly see how often and how much a software developer is contributing to a project. For example on GitHub we get a graphical perspective of a user’s commit activity – a developer’s punch card.



samdark's Open Source Contributions

16 contributions on March 5th 2013

Summary of Pull Requests, issues opened and commits. Learn more

760 Total
May 16 2012 - May 16 2013
Year of Contributions

13 days
May 04 - May 16
Longest Streak

13 days
May 04 - May 16
Current Streak

[4]

We can also gain insights into particular code bases for example how many people contribute to it and how frequently. In the GitHub archive we can find **event data**. Which is data on when and by whom comments were issued, repositories were starred or forked, pushes/pulls were made and even when merge conflicts arise. This could be used for example To gauge the popularity of a developer's projects in the community, you should check how many people have starred or "forked", forking or starring code indicates that other developers see it as valuable.

We can see the contributions that a developer has made to other projects and their own on the site and from that we are able to **learn about the skills** a developer possess and has used before. You can gather the information

[4] Image from here

pertaining to the coding languages the person usually writes in for example "rails" should be visible if a person has contributed to a Ruby on Rails project.

3. THROUGH TIMING

We can learn from **meeting lengths**. Whether or not a meeting is finishing on time conveys the circumstance of a team. If it finishes on time it can show that everything is running smoothly.

We can also look to **features completed on time**. Client satisfaction is achieved by bringing the client's project to life. The only way to bring a project to life is to implement it's features so a good show of progress is to keep track of how often and how well features are completed on time.

Another interesting measure is the **time spent fixing bugs**. We should note here that a lot of time spent fixing bugs is not a sign of a poor developer. It could be a result of technical debt or poor legacy code that are in need of maintenance, in that case a programmer who spends a lot of time fixing bugs in code would be considered quite valuable. Conversely when the bugs have been introduced in newer code and crop up quite frequently this would be a sign of a poor developer in a team as it shows us that bad code is being produced often.

4. THROUGH TESTING

Code coverage is a measure of how much of a program is run when testing is carried out. This is an essential measurement if your team is using Test Driven Development. One hundred percent code coverage is the ideal, this means that every line is executed at least once during a test. We can could use this metric to say with certainty that a particular software engineer has thoroughly investigated and tested their produced code.

We can look at the **tests a programmer writes for themselves**. Here we are able to tell how thoroughly a programmer has thought through their solution. If they attempted to account for edges and if their code actually solves the problem given.

In industry once a developer is satisfied with their solution it moves into the quality assurance stage. Where it is tested by a different group of testers who if they discover a problem will flag it with the developer and send it back to be corrected. That is known as quality assurance kickback (**QA Kickback**). A good indication of code quality and the developers efficiency would be to look at how often their code gets sent back to them and why. It would present an issue if code was being sent back to the same developer for the same or similar bugs often.

5. MESSAGING PLATFORMS

Slack and Microsoft teams are the most commonly used messaging and team management/communication platforms used in the software development community, especially in industry. Slack is a cloud-based proprietary instant messaging platform developed by Slack Technologies and Microsoft Teams is a unified communication and collaboration platform that combines persistent workplace chat, video meetings, file storage, and application integration. Other companies such as Facebook have designed their own internal messaging system. In industry the company has access to these messages and has the opportunity to learn from them. So what can we glean from messages between teams? From this we gain **insight into** how a **team** functions as a unit and indeed how a person functions in a team. We can use available software to identify the tone and sentiment of messages sent between team members to see how well they are working together. High friction messages especially close to a deadline could indicate a misstep in the development.

6. PREFORMANCE MONITORING

We could turn to an alternative solution and choose to record the software engineer as they work. This method has been previously adopted in the academic realm when conducting studies about developer performance and in industry. Through **monitoring a developers every action**, e.g. keystrokes, screen monitoring, it would enable management to perform a full review of their performance. The downside of this is that it would require a large time commitment, particularly in large organisations or teams. Some managers may consider this a worthwhile endeavour.
We could also monitor the **agile process metrics** to focus on how agile teams make decisions and plan. These metrics do not describe the software, but they can be used to improve the software development process.

I have discussed a large number of potential sources of measurable data for assessing and analysing the efficiency and quality of a software engineer’s performance. In industry it would be a decision of management to decide which metric they deem most appropriate. I will now go on to talk about how we can further analyse some of this data.

What do we need to avoid?

The dangers and trap falls that go hand in hand with analysing this data.

When done effectively these measurements enable better assessment and prioritization of problems within projects. In the introduction I touched on Spiegelhalter's example of how to go about counting all the trees on the planet, how they needed clear definitions of what they were measuring. Following that software metrics should have the following characteristics:

- Simple and computable
- Consistent and unambiguous
- Easy and cost-effective to obtain
- Able to be validated for accuracy and reliability
- Relevant to the development of high quality software products

This is to help ensure that the measurements don't give us false statistics. Furthermore data needs to be fully analysed before decisions should be implemented from it's findings. This could lead to frustration with management for not really understanding the issues. Martin Fowler, a renowned British software engineer and author would argue that "false measures only make things worse", in that aspect I have to agree with him. He goes on to say that "This is somewhere I think we have to admit our ignorance"[5].

With this I partially disagree, I feel that the notion of failure is definitely not a reason to give up. I do however think we need to be conscience of our ignorance and in doing that we could better learn from and improve based on our data. To do that this I would suggest being mindful of the following:

[5] "Cannot Measure Productivity" by Martin Fowler

GAMING OF THE SYSTEM

We need to be aware that statistical padding is a problem. The Hawthorne effect is an unavoidable bias that results in research conducted on humans that the researcher must try to take into account when they analyse the results. Subjects are always liable to modify their behaviour when they are aware that they are part of an experiment, and this is extremely difficult to quantify. All that a researcher can do is attempt to factor the effect into the research design, a tough, proposition, and one that makes social research a matter of experience and judgement. A study to establish whether cerebellar neurostimulators could mitigate the motor dysfunction of young adults with cerebral palsy found that the Hawthorne Effect adversely affected the findings. Objective testing showed that all of patients reported that their motor functions improved and that they were happy with the treatment. Quantitative methods however, showed that there was little improvement, and researchers invoked the Hawthorne Effect as the main factor skewing the results. They believed that the extra attention given to the patients, by the doctors, nurses and therapists, was behind the reported improvements in the initial study.

It is the Hawthorne effect that would skew the results of data gathered on tests above such as lines of code. Bill Gates once stated that "Measuring software productivity by lines of code is like measuring progress on an airplane by how much it weighs"[6] it a word it is redundant. Allow me to elaborate on why I agree so wholeheartedly with this statement. Engineers could create code that is artificially longer in order to skew the data of such a metric in their favour. Achieving this would be as easy as adding extra spacing in functions and using longer versions of certain productions to inflate their line count. This would lead us to a situation where our attempt at measuring productivity would lead to a decrease in the quality of code as often having a shorten solution is preferred as they are easier for other programmers to work with and if necessary improve in the future.

READING THE DATA CORRECTLY

The next thing we need to take into account when analysing these statistics is that we have to let the data draw the conclusions and not to look for a conclusion in the data. We cannot assume that because we choose certain

[6] Cs3012 webpage

metrics that they and only they convey the complete productivity of our company/project/team.

I look to basketball for an example. [7]In a single game a team worked their way to 51 offensive rebounds. At the end of the game the coaches could not figure out how they could lose a game where the dominated the offensive glass so much. The answer is easy, to get 51 offensive boundaries a team has to miss a lot of shots. The team lost because they shot horribly. The conclusion is not to harp on offensive rebounding but to find better shots, shots which they can make.

CORRELATION DOES NOT IMPLY CAUSATION

A correlation is a mutual relationship of connection between two or more things[8]. Causation is the action of causing something[9]. When looking at the measurable data we have gathered and attempting to identity positive and negative areas in the development process it is extremely important that we do not confuse correlation and causation. For example, if a team's productivity dropped at the same time that a new member joined the team, it would be unfair to immediately blame the new team member. There could be a variety of other reasons that you would ignore correcting if you do link correlation and causation unjustified.

LOOKING SOLELY AT NUMBERS

Software metrics can be extremely seductive to management as complex processes are simply broken down to numbers. Obviously it is easy to compare numbers to other numbers and thus when a certain software metric target is met we can declare success. Conversely if we do not meet this software metric target we know that we need to work more towards achieving that goal. This can make production static.

Trends work in our favour here. They allow us to distinguish between recurring problems and once off issues. Is a certain developer underperforming? Are there issues with the VCS we use? Were the time constraints on this project

[7] USA BasketBall – Don Kelbrick – Why Statistics won't tell the whole story
[8] From Oxford
[9] From Oxford

too tight? Analysis of why the trend line is moving in a certain direction or at what rate it is moving will say more about the process. Trends also will show what effect any process changes have on progress.

Unfortunately in this case there [10]"is no single development, in either technology or management technique, which by itself promises even one order-of-magnitude improvement within a decade in productivity, in reliability, in simplicity." . In essence there is on silver bullet. However insightful these metrics allow us to be we have to be aware that they do not totally encapsulate or explain all productivity, all successes and all failures though they do help us to understand a lot of them.

The ethics behind measuring and analysing software engineering

It is important when considering undertaking something like this in an industry setting to consider the ethical quandaries that accompany it. We have to consider our employees, our production, our prosperity as a business and what doing this can contribute to society. The ethical issues that stuck out to me were:

IS GATHERING THIS DATA OKAY

Perhaps the first question that springs to mind when you breach the subject of measuring software engineering is whether or not collecting and using this data is okay?

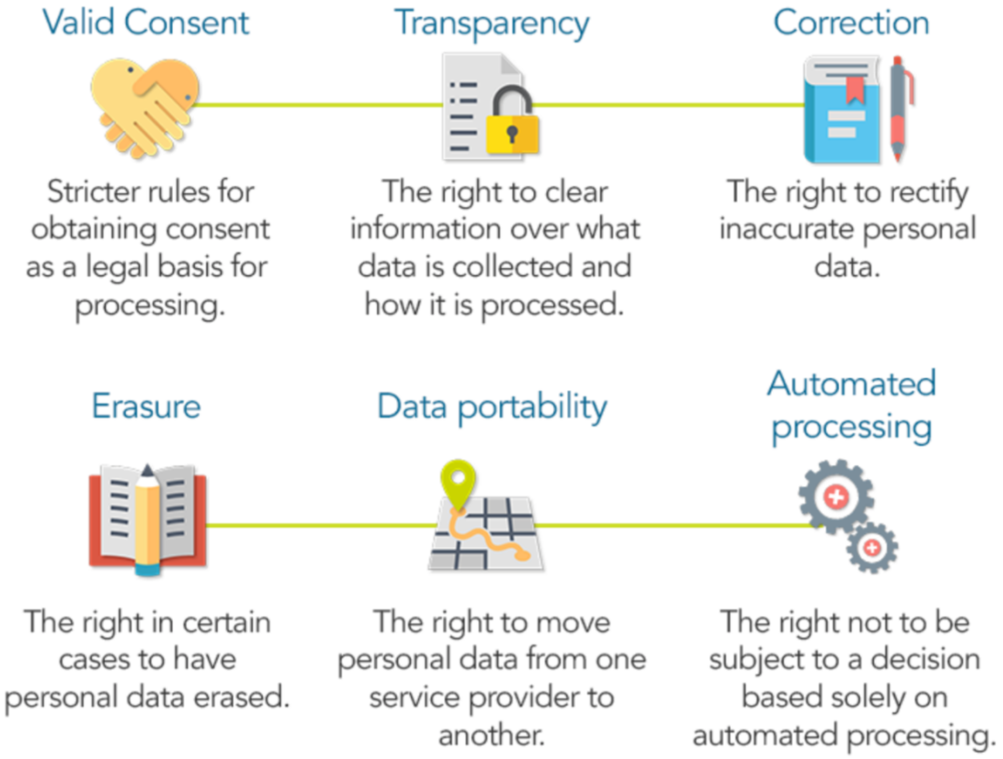
I believe an employer has the right to know how valuable their employee is to their company and business. I would even hope that in having data that was gathered intelligently and ethically and that was analysed correctly would give a clear opinion as to the assets to a company and even in some instances bypass personal biases. I do however think that there should be boundaries to the data collected. There is a **distinction between an employee's professional and personal data at work**. While data relating to their productivity is in my opinion fair game.; Data from their GitHub, LOC, percentage code, tests

[10] CS3012 webpage

employees write or meeting lengths. I think data that involves monitoring a more personal aspect of an employee's work life and work process should not be gathered by the company. For example, the notes they take at a meeting, the drawings and examples they use to help themselves understand a problem, the amount of time they spend sitting at their desk thinking rather than using their computer or even more personal data like bathroom breaks. This data is personal to the employee and I believe they have a right to expect it is kept private.

WHAT CONDITIONS ARE ATTACHED TO COLLECTING THIS DATA?

I think that first and foremost the employee has to consent to the data being gathered on them, I think it is the responsibility of the company to make it clear what data they are gathering. This may evoke the aforementioned Hawthorne effect but I think the ethical implications of not informing the employee far outweigh the benefits of them not knowing. It would be a breaching the employee's right to privacy not to. I think that the employee should also be well informed on exactly what the data gathered is used for. They should have access to their own data in a read only capacity so that if any false data presents itself, they have the right to challenge it and if validated correct it. The European Union seems to agree with me on this.



Valid Consent

Stricter rules for obtaining consent as a legal basis for processing.

Transparency

The right to clear information over what data is collected and how it is processed.

Correction

The right to rectify inaccurate personal data.

Erasure

The right in certain cases to have personal data erased.

Data portability

The right to move personal data from one service provider to another.

Automated processing

The right not to be subject to a decision based solely on automated processing.

Above is part of the EU General Data Protection Regulation Infographic on the privacy rights of an individual.

Furthermore in gathering this data it is my opinion that the company would have to take the necessary technical and organisations measures to safeguard the data.

1. Who has access

This is referring to the measurable data we would gather from the various sources discussed above. Who should be allowed to see, analyse and learn from this data.

As mentioned above I believe the employee themselves should have read only access to their own data. The company should allocate certain people to the gathering and analysis of their employees data either internal or external but I do feel that is the companies responsibility to ensure that that data is secure. This is because since the company chooses to gather this data in order to improve their software development model they have a responsibility to ensure their employee's rights are being met. The result of a data breach in this aspect is nothing short of chilling. One possible negative outcome of a data breach would be an employee with a less active lifestyle suddenly facing higher insurance premiums, if their insurance company was to get a hold on their activity data. Marketing companies could end up circulating the employee's personal information, as they seek more data for targeted advertisements. The level of infringement of the software engineer's privacy of such an event would be catastrophically high.

2. Looking out for the well-being of employees

Not only would it be unethical to constantly monitor employees from a well-being perspective but it is also just not smart. To again draw from a radical example if we look at people under an authoritarian rule, they are under constant surveillance. [11]According to Joshua Franco, a senior research advisor and the deputy director of Amnesty Tech at Amnesty International. "The fear and uncertainty generated by surveillance inhibit activity more than any action by the police. People don't need to act, arrest you, lock you up and put you in jail. If that threat is there, if you feel you're being watched, you self-police, and this pushes people out of the public space.". I'm not arguing that employers will lock you up for taking too many trips to refill your water bottle but I'm

[11] Vice.com – Kaleigh Rogers – What constant surveillance does to your brain

saying the constant monitoring of employees will leave them in a state of unrest and lead to a stressful work environment. According to an article published by Forbes "employees suffering from high stress levels have lower engagement, are less productive and have higher absenteeism that those not working under excessive pressure.". Looking out for the employees well-being is in the firms best interest as increased productivity is the main goal of the measuring of software engineering.

Conclusion

In conclusion the measuring of software engineering is a practice that companies should undertake to improve their productivity. They should make use of data available to them about an employee's work to both help the employee improve and to elevate the companies approach to software engineering as a whole. So long as the data gathered ethically and sensibly and is analysed in an informed and practical way I see this as a way for the industry to evolve in a positive way.