Mark Noone
15319898
13 November 2017

# Software Engineering

"To deliver a report that considers the ways in which the software engineering processes can be measured and assessed in terms of measurable data, an overview of the computational platforms available to perform this work, the algorithmic approach available, and the ethics concerns surrounding this kind of analytics"

## Introduction

In this essay, we will be discussing the question above. We will be breaking this question down into the key components that it contains, these being what measurable data can be taken from a software engineer with regards to his work, his communication and participation within a project team and finally his communication within the software engineering community by using sources such as Quora or Stack Overflow. Upon figuring out what data is needed, we will then explore the methods of computing results with this data. Here we will see what services are on offer to companies and engineers that allow both parties to see their performance statistics and how to better them. Following from our discussion of data computation, we will then discuss the possibility of computational intelligence within the software engineering industry. We will see whether or not we can use computational intelligence to find data that is worth collecting and whether we can identify good engineers versus bad ones. We will then finish our discussion around the ethics of all that we have discussed. It is important to see what we feel as both professionals and humans towards this collecting of data. We will address questions such as: Is it right? Should we allow our employers track as much as they can during work hours? What other information can be determined about us with the small data sets our employers can gather? This is always an interesting debate and one can find multiple opinions on each of the questions raised depending on those that are consulted.

## Measuring Software Engineering Processes

To begin our discussion, we must first know what exactly a software processes is. The following is our definition of a software process from our module textbook "Software Engineering" by Ian Sommerville, "A software process is a structured set of activities required to develop a software system." With this definition we now know we must find some data that we can use in order to track and measure the very activities that software engineering processes consist of, thus we must produce a data taxonomy.

Developing our taxonomy first begins with identifying the data needed. We will be using some of the core categories mentioned from the Personal Software Process (PSP) and Team Software Process (TSP), both created by Watts Humphrey. To give a small description of the PSP, Wikipedia describes it as the following. "The Personal Software Process (PSP) is a structured software development process that is intended (planned) to help software engineers better understand and improve their performance by tracking their predicted and actual development of code."

The TSP then builds on this idea and then "provides a defined operational process framework that is designed to help teams of managers and engineers organise projects and produce software products" of any size or difficulty. Humphrey designed both the PSP and TSP with his aim being to understand, monitor and manage the software engineering process because this is where he found big and small organisations or individual engineers encounter the most serious difficulties. In line with Humphrey's aims we will therefore be using ideas from both the TSP and PSP to help us identify and justify our data taxonomy. It is however important that we select measurable data from rich sources such as GitHub or a companies private tracking systems. These sources provide data, such as commits, bug fixes or lines of code produced which are unbiased and not influenced by the presence of those looking to record the data.

In order to produce our Taxonomy, we first must find our categories of data. We have found that the basic unit of a software process is the fundamental activities in which any software process consists of. These are specification, design and implementation, validation and evolution. Using these fundamentals as our basis, we must find categories of data to which we can accurately measure and compare various different software processes. Our core categories that contain the data that we'll be retrieving will be:

- Engineer Productivity
    - A necessary category that will allow us to measure the productivity of those engineers that are working on a given task or activity. With this, we are able to measure how each developer and teams of developers respond to the various different software processes.

- Code Quality/Functionality
    - This will allow us to measure the output quality of the engineers being measured. It follows on from the productivity as mentioned above however will test whether the engineers are producing good quality software.

- Activity Time
    - This is the measurement of time taken to complete parts of an activity. This can be time taken to complete a development task, bug fixes etc ..

A table specifying the data metrics measured is shown below:

| Category | Metric |
|---|---|
| **Developer Productivity** | Lines Of Code Produced - Amount of lines of code contributed to the codebase. |
| | Commits Made - The amount of commits made from a developer to the codebase. |
| | Active days - Days in which engineers have contributed to the codebase. |
| | Bugs Fixed - Number of bug fixes within the codebase from a developer. |
| | Messages Sent - Messages sent to team project contacts and chat rooms. Shows integration and good communication with the team. |
| **Code Quality/ Functionality** | Code Churn - percentage of a developer's own code that requires editing to their work soon after it has been created. |
| | Bugs Recorded - Amount of bugs recorded in developed software from users, testers or developers. |
| | Unit Test Coverage - Test coverage is the degree to which unit tests cover and test all possible outputs to the software codebase. |
| | Technical Debt - The extra development work that arises when code that is easy to implement in the short run is used instead of applying the best overall solution. |
| | User Reviews/Ratings - This is a direct star rating and possible written opinion from the user to how well they feel your product performs. |
| **Time Spent** (**Minutes**) | Lead Time - Time taken to go from specification to validation and evolution. |
| | Cycle Time - Time necessary for a change to be pushed to production. |
| | Time spent on fixing bugs - Amount of time spent on making fixes from previously reported bugs |
| | Time spent on refactoring - Amount of time spent on refactoring old code for software evolution. |
| | Open/Close Rates - Time taken to close a bug from time recorded |

With these measurements we are trying to identify and find out:

A. How well our developers are motivated, communicating and writing software within a given software process. By monitoring the likes of active days, commits made, messages sent etc, we can accurately judge an engineers integration with the team and time spent on developing. Some software processes can work for developers and others may not. Being able to compare the various processes among developers will allow the software managers to clearly see which processes suit the majority.

B. The quality of code being produced. Code quality is a huge part of any software engineering. When code is written we expect it to hold few bugs and last as long as the codebase relies on it. This is why we monitor metrics like unit test coverage, technical debt or bugs recorded on an engineers work. It is also not enough to simply monitor metrics such as lines of code produced or commits made and nothing else. This is because a developer could be writing lines upon lines of code and look productive, but when we add in metrics such as the code churn of a developer we can quickly see the amount of lines that are produced which are of good quality.

   Being able to see these metrics will allow us to measure the development of quality code will allow us to compare various different processes and determine which is the best for our developers.

C. The time spent in completing tasks and activities within a software process. This can be seen by the various time measurements we have mentioned above. Lead time allows us to gain an understanding in the timeframe that it takes to complete one "sprint", as we'll call them, straight from the start of the process (specification) through to the end of the process (validation and evolution). Cycle time allows us to view how long it will take to get a necessary change, be it a feature or a bug fix, straight to production and into our users. This is an important metric as if this time grows it could turn users away from using a possibly "broken" product. If this happens, a change in process may be needed. Our other time related metrics all allow us too view the amount of time developers spend refactoring or fixing code.

   Obviously time is a tight resource in software engineering and being able to better estimate and understand time frames will allow us to compare the software processes on time alone, but with also using the other metric categories mentioned, we will be able to gain a complete comparison.

With this clear data taxonomy created and our metrics defined and justified, we will now discuss the computational platforms available to compute the results we need to compare our various different software processes.

## Computational Platforms and Monitoring Tools

When it comes to computing good data that we wish to use in our comparison, there are a huge range of options available for companies. They can go down the road of using vast computing power with the likes of Amazon's AWS services to using cloud Saas (Software as a Service) platforms, like code climate[1], to calculate, monitor and determine code bases and engineer productivity. In this section we are going to look at a mixture of the platforms and tools available to developers and software companies that can monitor both software engineers and the software they create.

We'll first begin with Amazon and their cloud-based web services platform. Here Amazon have provided a huge amount of computing power available for companies to use in any way they wish as a service. A company can completely build their platform or monitoring tools as they wish without having to invest into any hardware infrastructure, which can often be an expensive part of any business. These computers are completely available to companies for a very good price. However, it is to be noted that this approach can often be much harder and more time consuming for companies to set up as they must invest time into researching the best monitoring solution for them and, if none exists, sometimes build their own solution. We will not be discussing building a companies own solution but instead will mention some of the cheaper monitoring solutions that are set up and configured, some of which are open source.

The Amazon web services are not enough to monitor systems and software applications. They must also be setup up with some kind of monitoring application as well. There are many different open-source monitoring solutions that each specialise in their own area, however we will be showing off three of some of the top solutions available and how they work in parallel with a software engineer's application. Tools like Nagios[2] or Zabbix[3] are open source tools available to monitor servers and any other server-side environments that applications are executed on. They work via a server-agent architecture. This means that there is a centralised server that will regularly poll what we call "agents" for information. As we can guess, agents are the environments that we wish to monitor. These agents are configured by the engineer in such a way, that they will report back any specified information to the centralised server. This one centralised server then will keep track of many different agents. This then takes care of any server-side monitoring.

Following on, engineers can implement further client-side Real User Monitoring with frameworks like Boomerang for web apps or Microsoft's application insights for desktop applications. All of these solutions generally implement a similar server agent architecture. With companies like DELL, Salesforce, ICANN, Orange implementing these monitoring solutions, it is clear that there exists value in keeping track of all this information. However, as

mentioned previously it can be tiresome and an inefficient use of time to develop all of this monitoring infrastructure, especially for the like of startups where time is one of the greatest resources.

Another path for companies is to subscribe to services that monitor all that we have mentioned for you and more. Companies like Code Climate[1] are companies that specialise in developer and codebase monitoring. They themselves offer a service that pulls down repositories and breaks it down into digestible chunks, making clear graphs and showing managers exactly how their projects are going with regards to productivity and testing. This answers questions like who made what changes, when they made them, why they made them, is the unit testing for them good enough etc.. Code Climate's flagship users New Relic, Intercom, and Barracuda , whom are large respected companies, show that being able to monitor repositories alone and retrieve this kind of data show us that businesses are taking advantage of these SaaS monitoring platforms and feel they are of great value. Code Climate also provide their service for free to any open source developers and students.

Our main topic of this essay is to consider the ways in which the software engineering processes can be measured and assessed and here we talked of the tools and platforms that can allow this. Both the proprietary solution, through using the likes of AWS and the more streamlined subscription solution are both present for companies with some at a greater cost than others. Being able to see how the codebase is being grown and running (Code Climate) will allow companies to monitor our developers productivity and how much of their code is of good quality, while the more proprietary solution (AWS + Open-Source) will allow companies to monitor the performance and again quality of their software as it runs in production however more customised and catered to the companies needs. Both of these are of great value to companies and possibly a hybrid solution is the best however both on their own will allow companies to monitor how the various different software processes within their companies are performing with the various different developers.

We have all this data present and calculated and can determine what we have discussed above, however we will now discuss in the next section if we can use this data to a more individual level and possibly determine what a "good" software engineer is before and after he is hired in a company. We will attempt to discuss this in relation to both CI (Computational Intelligence) and non-CI methods.

## Algorithmic Approaches

Software companies want to be as efficient as possible, producing high quality software that rarely has problems. Measuring software project success and developer productivity is useful information for individuals and hobbyists looking to spend their time wisely. This is also the case for companies, as they are always looking to spend their money wisely by keeping costs low and profits high. From the previous section above, we can now determine that companies are very interested in measuring the effectiveness and productivity of their engineers under different software processes. This to keep the main cogs of their business churning efficiently. We are now going to look at how we can use data from services like GitHub as we discussed in the first section where projects, engineers, and changes are readily open to inspection, to then in turn figure out how we can possibly use this data along with computational intelligence to determine those engineers that are best for the business, both in regards to employees and engineer candidates.

Computational intelligence is an offshoot of artificial intelligence that is becoming a growing field in the world of computer science. Typically, branches of computer science are not defined as typically other sciences are. Instead, they slowly grow and develop as a cluster of common interests. . In artificial intelligence, these interests generally focus on problems that usually only humans and animals can solve, thus these are problems that require some form of intelligence. Computational intelligence is today an umbrella for three core technologies. These are:

- Fuzzy logic - which enable the computer to understand the likes of natural language.
- Artificial neural networks - which enable a computer system to learn experiential data by operating similarly to a biological system.
- Evolutionary computing, which help in dealing with uncertainty and imprecision. These are largely based on the smaller processes of natural selection, learning theory, and probabilistic methods.

Computational intelligence has no concrete foundations and is known more as a group of techniques used to achieve non-algoritmizable, lower cognitive, more practical problems than a solid branch of science. These days companies are using CI to do everything from service optimisation, to building and improving manufacturing and supply chain management. The idea of using CI in todays world has definitely been tapped and with this in mind, we can therefore see how companies and businesses may possibly be able to use statistics and data from their engineers to analyse and determine the “perfect” or “good” engineer all using CI. Now as we mentioned before, there are three main techniques in CI. These are fuzzy logic, artificial neural networks, and evolutionary computing. For our application, neural networks seems like the avenue to go down as it involves manipulating hard datasets and adjusting our

neural network to learn all about the perfect engineer. This is exactly what we want, however neural networks need a key metric or metrics that can be used in order to tell the network of it was "right" in judging the best engineer. When building machines to play games for example, this key metric would be something along the lines of a score within the game. Here, we have an array of valuable metrics as discussed in the first section. With industry leaders like Facebook and Microsoft and most startups like Code Climate, who find themselves in the area of assisting companies in high quality software, claiming that their primary focus is to deliver high quality code, it seems clear that metrics such as code churn, lines of code produced and technical debt all along side some form of time measurement as defined in the first section would be the most appropriate metrics for our hypothetical neural network. However, it is to be noted that companies can change these metrics around based on their ideas and principles as long as there is a dataset to measure. This is the beauty of neural networks and can thus prove that some form of computational intelligence can be used to determine what companies feel the "best" engineer is.

To bring another element into the discussion, one should note that there are two different types of learning algorithms. These are unsupervised and supervised algorithms. The type we are currently discussing is supervised as we know exactly what data we have and what we want to see as our output (our "best" engineer). In unsupervised algorithms, this is not the case, not only does the algorithm not know anything about the data it's operating, but it also doesn't know anything about the output we want. It is up to the algorithm entirely to determine the type of data it is shifting through and recognise any patterns that may or may not exist. From this, the algorithm will then output something that it deems the best. Because we cannot control this output, I believe companies would not deem this type of algorithm beneficial. Companies have a very particular idea of a "good" engineer and I believe they would want a neural network to work towards their idea rather than determining one for itself. To finish off, computational intelligence is not the only avenue that a company could go down to determine the quality of engineer they wish to hire or reward. Some companies can use non-CI algorithms that are supplied by some of monitoring and repository companies like Code Climate, however, these give more of a brief history on a developer and how well they work rather then trying to judge if they'll be a good fit for a company. This is helpful to companies now but using Computational Intelligence to analyse and identify patterns Mong developers could prove to be a greater asset for companies in the future.

Both CI and non-CI approaches exist in the scope of determining "good" engineers. With current industry leaders constantly looking for new methods to test and rate potential new engineers and even determining the quality of the current engineer, the methods discussed above have the potential to become the standard in the computing industry. However, is this right? Should we become so used to being monitored 24/7 that companies get away with determining whatever they wish about us? What happens if we have a bad week or month, do

we get sacked? Finally who owns the data that we provide, ourselves or our employers? These are all questions that surround around the ethics involved with monitoring and comparing engineers and it's what we will discuss in the next section.

## Ethics

We will now be discussing the ethics involved in monitoring software engineers. This paper outlines a method of monitoring and measuring the software engineering processes with clear measurable data. We will try to discuss and debate what we have mentioned so far into account.

There is always a detailed ethics debate surrounding technology and people. For some, it is a very strong, passionate debate as people feel we are allowing technology and computing companies to hold too much information on us. With this power, they are figuring out more then we know about ourselves using the huge amount of data available and then using that to their advantage with the likes of targeting advertisements and marketing campaigns. For other techno utopians, this world of companies determining what a person wants instead of themselves as previously described has always been a good thing. Who wouldn't want to be told what to buy? That's fantastic that you know what I need or what movie I want to watch tonight. For these people, the benefits of companies being able to calculate this kind of information only has benefits and they want nothing more than to keep it that way. This is very generally the two sides of the ethics debate within the technology scope. Both with very good arguments for their case. However, for this discussion we are talking about software engineers who are more commonly known to be on the techno utopian side of the debate. This is to be expected as these are the very people creating the platforms that we all use, so they know how it feels to be following data guidelines and they can see how much easier it makes the life of an engineer by not worrying about ethics. To follow on, they know the exact benefits that companies can bring to people with all this data at their fingertips.

With this in mind, we can see how most engineers don't mind about companies monitoring them. In their minds they believe they are doing great work and if they are not, they want to know why so they can soon be doing great work. After a degree in computer science, a career in software engineering is not out of the ordinary. I could well personally be regularly monitored in my future workplace and constantly be checked against both my colleagues and most likely past and possibly future employees. With this, in my opinion, comes both positives and negatives within the workplace. Some benefits we could see is increased productivity within the workplace with this almost competition among colleagues, and people feel they are being rewarded for good work. On the other hand, if companies are not willing to fully invest

themselves into the idea of monitoring their employees, we could see engineers becoming unhappy feeling like they are not being treated like a valuable asset and contributing regularly and positively to a codebase. This could then lead to people leaving an organisation and could even cause outrage among the software engineering community. Thus I believe that if a company was to begin some form of monitoring it would have to be a fully fledged and well invested solution that is accurate and transparent among the engineers it monitors bringing trust and a feeling of value into the workplace.

If this was to happen, I believe it would greatly increase productivity in the workplace due to the almost gamification of work. This has been a clear tactic to motivating workers in software engineering groups with many books written on the topic. A summary of the general theory of gamification as a motivational tactic can be found on Medium[4]. Another benefit I feel monitoring would bring to the workplace is engineers would be rewarded for their performance and contribution within their work. I am a firm believer in rewarding employees for their work completed, especially if it's good quality. If you are bringing value into the business you should be rewarded for that contribution and more so then the employees who have not contributed as much. Bringing a form of monitoring into the workplace as an aid to determine and motivate employees to work harder and of higher quality I feel would make software engineers happier. They will take higher pride in their work and would be rewarded for such, allowing them to enjoy coming into work each day. These are some of the benefits that I feel a monitoring solution would bring, however one can also note some negatives with the idea of monitoring.

A huge part of implementing a monitoring solution in a company, is how seriously they take the responsibility involved in implementing such a solution into their workplace. If a company was to be irresponsible, I believe it would have a hugely negative impact in the workplace. This is because so much of how a company carries out a monitoring solution effects the trust and happiness within the work environment. A company good not be recording data accurately enough leading to rewards for undeserving people or calling out poorly on hard working employees. Therefore, a negative for implementing a monitoring solution must be how much trust and responsibility you must place on the employer. Another negative with engineer monitoring a feel would be the possible loss of good talent to the software engineering world. Software engineers are in very much demand nowadays and companies are crying out for good programmers looking to train them into successful engineers. If it became a standard for companies to implement these kind of monitoring solutions, it will inevitably turn people good talented people away because they don't feel comfortable with the idea of being watched during work hours.

Their also comes a massive debate over data ownership. You are producing a mountain of data for your employers that is bringing more value to their business everyday. They have

invested time and money into this monitoring solution regularly, improving it to make sure it suits the business at hand. With all of this work put into getting this data, the company is right to think that the data is owned fully by them, however an engineer is the very source of this data. They should feel that they have some part of ownership to the data they produce. This is a hot topic for discussion, however I personally feel that the data is owned by the company that you work for. You are employed by the company to work under the companies conditions as you agreed to on employment. In my mind, it is not only your job to produce the great code that you were hired to do, but also to produce this data for a company. However, I do believe that an employee should possibly be rewarded a bit more for this extra work. Alongside this, I believe if monitoring solutions where to be implemented into a work place, a company must be fully required by law to fully and accurately describe what data is being recorded, when it is being recorded and for what purposes. This will bring complete transparency and trust between the employer and employee, thus keeping the active monitoring of employees positive and valuable.

We have discussed very briefly here the ethics involved with monitoring engineers and the ups and downs included with it. Following on we discussed all about the data ownership and some of the responsibilities I believe companies would have to take on if they were to implement a monitoring solution. Monitoring I believe can be a very positive thing in a software engineering team, as long as it is accurate, rewards good work and only exists during work hours, making this non invasive and non distracting for workers. On the software company side, it could allow companies to measure everything about their software processes from the individual to the team, allowing them to figure out as much as they need about their business assuring that the software development cogs are as efficient as possible. Thus, making a monitoring solution a positive addition to the workplace, providing it is carried out fairly and responsibly.

## Conclusion

Throughout this essay we have considered many of the ways in which the software engineering processes can be measured and assessed in terms of measurable data, the computational platforms and tools available fo companies, the algorithmic approach available and what they can do for companies, and the ethics concerns surrounding this kind of analytics. It has been a small paper on four very large topics that could be read, researched and written about for a lot longer than this paper has. It is a brief summary on some of the thoughts and benefits along with problems that monitoring engineers can bring to a work place. It is an attempt to provide a baseline and to trigger insight and curiosity among it's readers. In my opinion, I believe that monitoring our future engineers definitely has good ideas, benefits and as a possible future engineer, I would be interested and all for any company fully invests into the idea, as long as it is done so responsibly.

## References


1. Code Climate, an American Codebase Monitoring Startup, https://codeclimate.com/
2. Naigos, Description goes here, https://www.nagios.org/
3. Zabix, Description goes here, https://www.zabbix.com/
4. Article on the Gamification of software engineers, https://medium.com/gameful-design/does-gamification-work-in-the-software-development-process-76780e49e545