



Trinity College Dublin

Coláiste na Tríonóide, Baile Átha Cliath

The University of Dublin

Measuring Software Engineering Report

Software Engineering - CSU33012

Prathamesh Sai

3rd year Integrated Computer Science

Student ID: 19314123

SCHOOL OF COMPUTER SCIENCE AND STATISTICS,
TRINITY COLLEGE DUBLIN

Contents

1	Measuring engineering activity	2
1.1	Frequency of commits	2
1.2	Code complexity	2
1.3	Size of commits	3
1.4	Number of forks	4
2	Gathering and processing data	4
2.1	GitHub API	5
2.2	Team collaboration tool usage	6
2.3	Cloud services	7
3	Computations on software engineering data	7
3.1	Simple counting	8
3.2	Expert systems	9
3.3	Machine learning	10
4	Ethical issues about personal data processing	10
5	Conclusion	11
6	References	11

1 Measuring engineering activity

As software engineers, it is important that we are able to monitor and measure our engineering activity. Not only to discover issues within the process of software engineering, but to also start a conversation about fixing those issues to improve the software engineering process over time.

It is clear to see that doing so would be of benefit to not only software engineers around the world, but also for the field of software engineering in general. There are a number of ways of measuring software engineering activity and I will now discuss a few possible examples one could analyze to measure software engineering.

1.1 Frequency of commits

The frequency of commits is a relatively straightforward method of measuring software engineering activity. It is logical to think that the frequency of the commits by a software engineer can give us an indication of their productivity and fluidity of their workflow.

If we wanted to assess the activity of a project, we know that the less active a project is, the longer the time between commits of individual developers (Kolassa et al., 2013). This acts as a straightforward way of measuring the activity of software engineers over a short and long period.

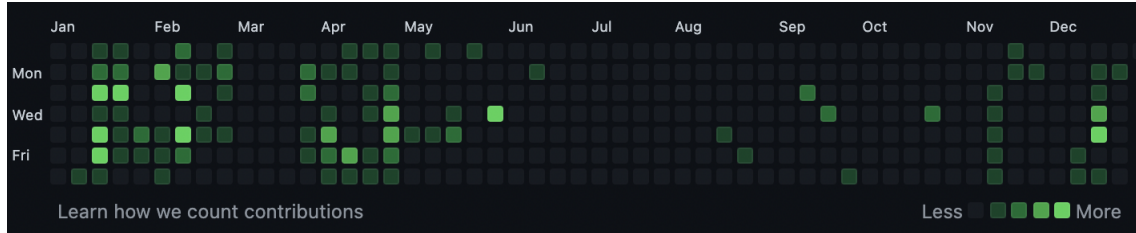


Figure 1: The frequency of commits can be used to measure software engineering activity. We can see the frequency of commits over the past year for my own GitHub profile above.

1.2 Code complexity

The complexity of the code written by a software engineer acts as a strong indicator to measure software engineering activity. Software engineering is a complex field. In a utopian world we would be able to solve everything with easy solutions. Although we are able to solve some problems like this, software engineers are sometimes required to develop complex solutions for complex problems.

As a software engineer writes code for a piece of software, software complexity is a measurement of the resources expended through the software life cycle (R.Gonzalez, 1995). These resources could include money for a company, but energy and time for the software engineers working on a project which allows us to measure their activity.

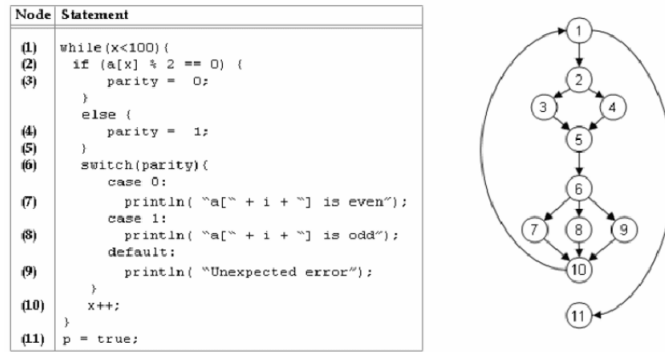


Figure 2: The code complexity of code written by a software engineer acts as a feasible method of measuring software engineering [3].

1.3 Size of commits

The size of commits is an interesting indicator of software engineering activity. As mentioned before, we can use the frequency of commits to measure software engineering activity, however the size of each commit is vital to the development of a software system.

Most developers follow the behaviour of committing small amounts of code frequently, to utilize the benefits of version control to roll back to previous versions if anything goes wrong. However, in a study it was found that many large commits actually modify the architecture of a system (Hindle et al., 2008).

This is important for any project, and acts as an important step in the software engineering process. This study found that small commits were corrective in nature, while larger commits were used to perfect a system which is more global in scope.

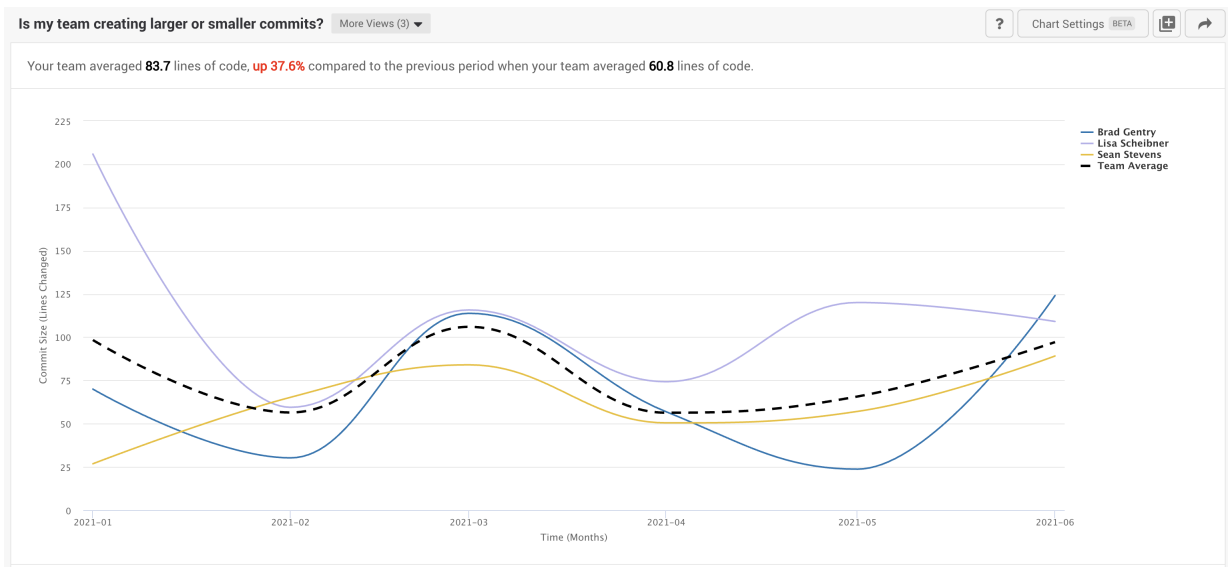


Figure 3: A team can measure the size of commits by each team member over time to measure software engineering activity [5].

1.4 Number of forks

The number of forks in a repository is a great way of measuring the effectiveness of a software engineering project. Software engineering is about working together. Nowadays, Open Source allows us to collaboratively work together with people from all around the world.

When there are more forks, there are more maintainers who are able to evolve a piece of software in face of change (Baudry & Monperrus, 2012). Therefore, there would be more people to improve the project over time, hence we can measure this software engineering activity.

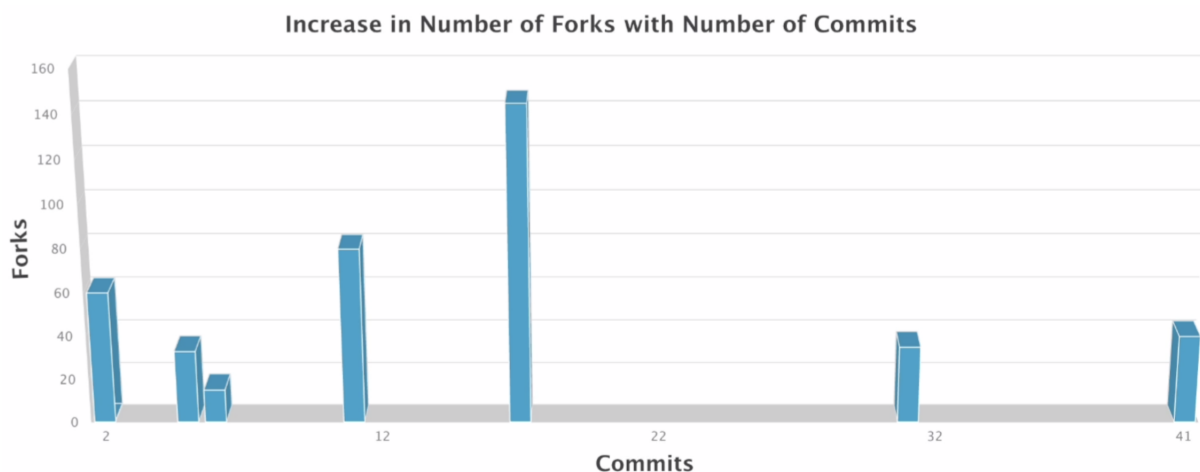


Figure 4: The number of forks in a repository is a great way to analyze the software engineering activity of a project. It shows how many people are copying the project locally to potentially contribute to it.

By measuring software engineering activity using the number of forks in a repository, we can use this data to gauge the collaborative nature of the project. Generally, the more software engineers work on a product, the more successful it can be. This is because of a range of factors such as more frequent bug fixes, more frequent updates, and more of a workforce on demand to fix any issues that arise.

2 Gathering and processing data

There are various platforms from which one can gather and perform calculations over data regarding software engineering activity. These platforms are essential as they allow us to collect important data to process and eventually reason with.

Among the various platforms from which we can gather software engineering data, I will mention a few in this section, and also discuss a platform which allows the previous two platforms providing data to compute heavy computations without hardware limitations.

Although we would only need platforms to provide more computing power if we are dealing with data on a large scale, it is important to discuss these options as many

companies with a huge number of employees may want to gather software engineering data for their company. They might need the necessary computing power to process all this data if they cannot do it on their own.

2.1 GitHub API

The GitHub API is a platform on which one can gather data regarding software engineering. Using Java, one could use the Eclipse EGit GitHub API Core 2.1.5 API to interrogate the GitHub API. You can gather information about the repositories of a user such as the number of commits, number of forks, number of watchers, and size in kilobytes. You can analyze each commit and see the author of each of them. Also, you could gather user data from a GitHub profile.

By using the GitHub API, you can gather all this data and store it in a database. If you use a cloud database such as MongoDB Atlas, you can then use the MongoDB Atlas API to access the your data. Then, you can use data processing to process the data you want, and visualize it in another application using Javascript libraries.

This allows you to access important software engineering data, store it in a database, and visualize large volumes of data with the help of algorithms. There are simple and more complex algorithms and data structures that can be used to gather software engineering data.

```
//Set data for size and forks graph.
this.sizeAndForks = [];
for(var j = 0; j < numberOfRepos*percent; j++){
    this.sizeAndForks.push([response.data[j].NumberOfForks,
        response.data[j].Size]);
}
const map = new Map();
for(var z = 0; z < numberOfRepos; z++){
    if(map.has(response.data[z].Language)){
        map.set(response.data[z].Language,
            response.data[z].NumberOfCommits +
            map.get(response.data[z].Language));
    }
    else {
        map.set(response.data[z].Language,
            response.data[z].NumberOfCommits);
    }
}

//Set data for language and commits graph.
this.languagesAndCommits = [];
map.delete(null);
for (const [key, value] of map) {
    this.languagesAndCommits.push({
        text: key.toString() + " (" + value + " commits)",
        values: [value],
    });
}
```

}

Listing 1: Algorithms such as simply iterating through all the data from a response and data structures such as maps to hold this data is important when processing data measuring software engineering activity.

2.2 Team collaboration tool usage

When working in a team, it is important to talk to team members about the complexity in your code or if you are stuck on a problem. Keeping in contact with team members regularly allows software engineers to work collaboratively and efficiently.

Team collaboration tools such as Microsoft Teams or Slack allow you to view usage reports or an analytics dashboard. By using these platforms, one can gather data about measuring software engineering. These usage reports come in various forms.

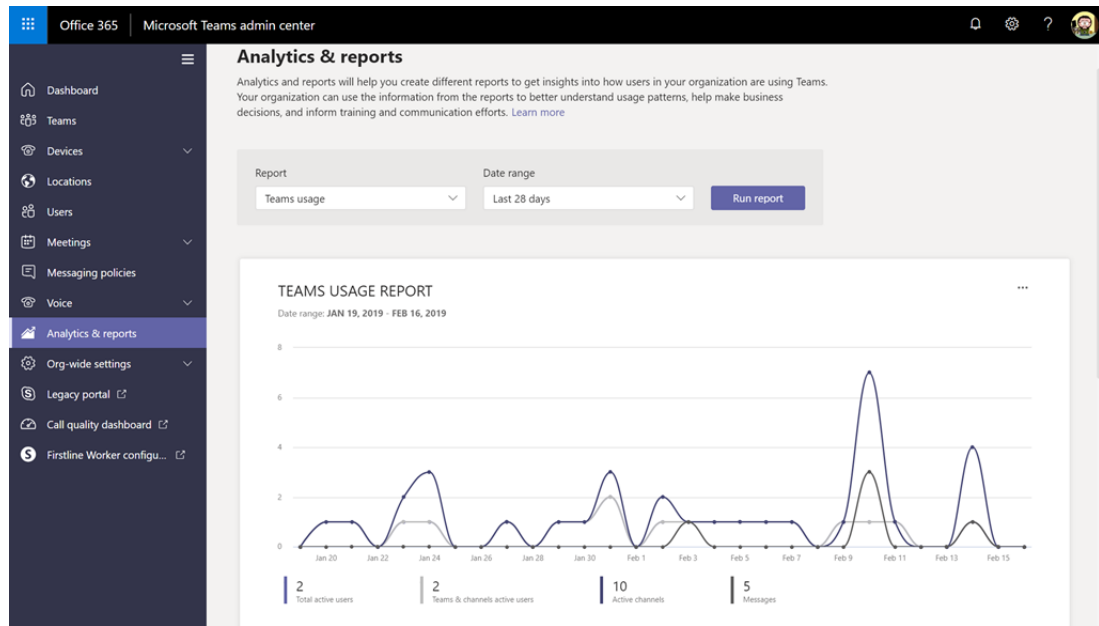


Figure 5: In Microsoft Teams, you can view analytics about the usage of particular channels, the active user in those channels, and it can process this data and visualize it for you [6].

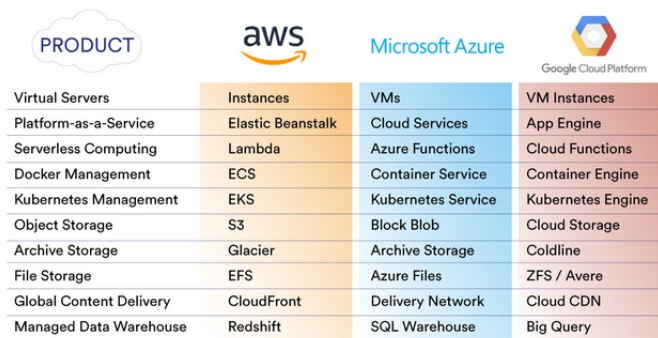
By analyzing these reports, one can gather data about software engineering productivity assuming the channels are purely used for software engineering discussions. One could then perform calculations on this data using a programming language to reason about the data given to them. It may find out things such as the lower overall productivity of software engineers on particular days of the week, or particular times of the day. Such information is important to know, hence this method is effective at reasoning about software engineering data.

2.3 Cloud services

I mentioned two ways of gathering software engineering data above, but what about processing this data on a large scale? Cloud services like Amazon Web Services, Microsoft Azure and Google Cloud allow us to use virtual servers to utilize their computing power to process the data we have at hand.

Without the appropriate computing power, we will not be able to process the data to comprehend it. With the help of cloud computing, one can utilize technologies such as Kubernetes and the cheap access to considerably fast hardware to compute and analyze data.

For example, if a company was trying to research software engineering productivity in a company of 100,000 employees, it would require the necessary hardware capabilities to perform heavy calculations on every employee to calculate overall software engineering productivity.



PRODUCT	aws	Microsoft Azure	Google Cloud Platform
Virtual Servers	Instances	VMs	VM Instances
Platform-as-a-Service	Elastic Beanstalk	Cloud Services	App Engine
Serverless Computing	Lambda	Azure Functions	Cloud Functions
Docker Management	ECS	Container Service	Container Engine
Kubernetes Management	EKS	Kubernetes Service	Kubernetes Engine
Object Storage	S3	Block Blob	Cloud Storage
Archive Storage	Glacier	Archive Storage	Coldline
File Storage	EFS	Azure Files	ZFS / Avere
Global Content Delivery	CloudFront	Delivery Network	Cloud CDN
Managed Data Warehouse	Redshift	SQL Warehouse	Big Query

Figure 6: Each cloud service has it's own benefits and provides slightly different services, but they can be used to provide adequate computing power to compute large computations on software engineering data [7].

3 Computations on software engineering data

There are multiple computations that could be done on software engineering data to profile the performance of software engineers. These computations are important as they allow us to extract vital information that we can use to improve and learn. For example, if we see that there is an overall productivity drop on particular days of the week, then one can arrange the working week such that there are easier tasks allocated on those days.

Another example would be if we wanted to do computations on software engineering data to figure out the overall work output of particular software engineers to rank their ability. By using a variety of data such as the ones mentioned at the start of this report like the frequency of commits, we can measure software engineering on a large scale to make it cost effective for companies to keep software engineers on track.

As you can see, there are many ways in which computations on software engineering data can be useful. In this section, I will discuss simple and complex techniques for data

computation. With these techniques, you can get interesting results which can eventually be used to infer theories about the software engineering process.

3.1 Simple counting

Simple counting is a simple concept, but it is one of the foundations of discrete mathematics and computer science. We can use simple counting to count various data such as the number of commits for each programming language used by a user, or the number of followers of a GitHub user. As trivial as it sounds, it is actually one of the simplest yet most effective computations we can do to analyze and reason about software engineering data.

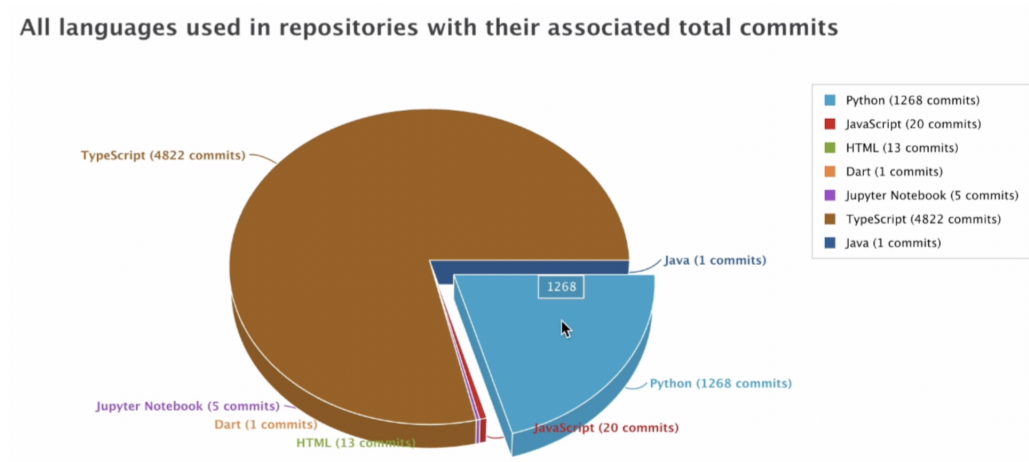


Figure 7: With simple counting, you can effectively count data such as the number of commits for different programming languages.

Another example of simple counting would be counting the followers a Github user has, and counting the users a Github user follows. With this relatively simple data, we can gather some background on the type of software engineer we are processing data from.

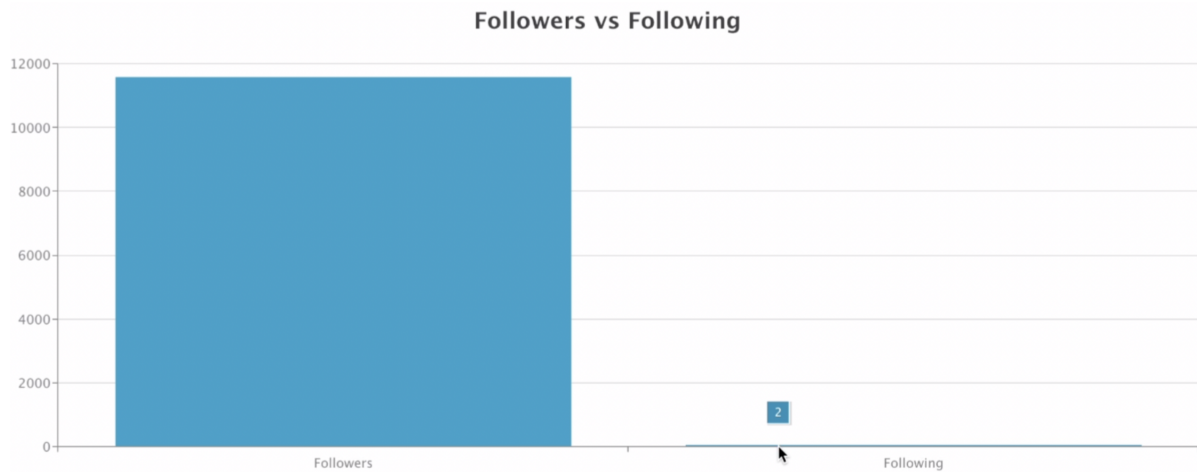


Figure 8: This GitHub user follows 2 people but has over 11,000 followers. We can infer that this software engineer is popular and potentially owns repositories that may be famous with many forks and watchers.

3.2 Expert systems

Expert Systems have built on many years of work in artificial intelligence (AI) on problem solving, and have become a commercially successful demonstration of the power of AI (G. Buchanan & Q. Smith, 1998). With expert systems, a computer is able to copy the decision making of a human using a knowledge base with an inference or rules engine and is able to come up with meaningful results. For example, there was an expert system developed to measure software engineering (He et al., 2003). This allows one to specify the data that was wanted from a project, and the expert system was able to process the data and visualize the findings.

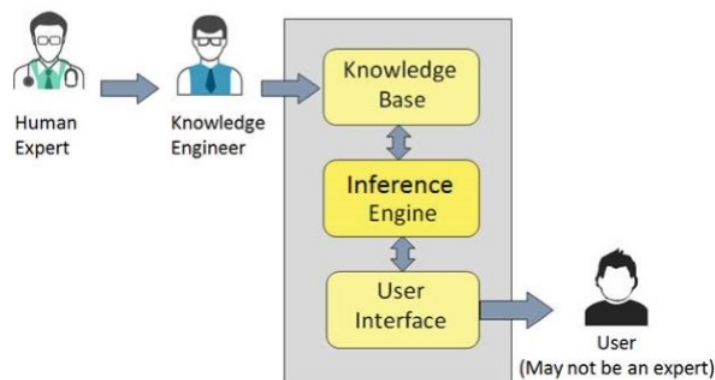


Figure 9: A example of how an expert system works [10].

Using expert systems, we can do computations on software engineering data even if we are not an expert on where the data is from. With the help of expert systems, we can compute complex calculations on software engineering data. This is a powerful tool as it

only needs a few parameters through an interface and we are given useful data that we can use to improve the field of software engineering for the better.

3.3 Machine learning

Machine Learning is the study of computer algorithms that improve automatically through experience (Mitchell & Hill, 1997). Machine learning is used everywhere nowadays. From image and speech recognition, to statistical arbitrage and predictive analysis, machine learning helps us to solve real world problems. In terms of measuring software engineering data, machine learning can be used to use past commit frequency to predict future data about a repository. By measuring past software engineering data, we can predict future software engineering data with machine learning algorithms such as linear regression. For example, there was an experiment done to measure software development productivity using linear regression (Helie et al., 2018). Productivity was measured using 2 factors: quantity and quality of code.

What if we use these factors to predict when a software engineer will be burnt-out in the near future? If we analyze their commit frequency and code structure using machine learning models, and accurately predict burn-out among a cohort of software engineers with 98% efficacy, what about the other 2%? There are many questions left unanswered with machine learning models to measure software engineering, as well as the ethical impact it may have. Does telling a software engineer they will be burnt-out in approximately 3 weeks a good or bad thing? If the model predicts correctly, then that it may have an emotional impact on them, but perhaps it prevents them from overworking themselves. If the model predicts incorrectly, it may still have an emotional impact on the software engineer, but may also lose time and money for the company who they work for.

4 Ethical issues about personal data processing

There may be concerns about the ethical, legal and potential moral issues surrounding the processing of personal data to measure software engineering as discussed above. Personally, I do not think there are many ethical issues surrounding the collection and processing of software engineering data. If anything, I think it would help the field of software engineering for the better. We would be able to come up with interesting theories based on human behaviour by evaluating software engineers. These theories will help us to improve the field for the next generation of software engineers. It does not necessarily cross any lines to analyze the performance of software engineers as they go about their work, as it does not impact their workflow. It acts as a method of improving the field that they work in. The practicing software engineer has great potential to effect the lives of others (Gotterbarn, 1991), and improving the experience and productivity for them shouldn't be seen as an unethical thing.

That being said, doing complicated computations using stuff like machine learning models to predict and alter the future of software engineers has it's own drawbacks. Doing this

complex data processing isn't always easy, meaning we aren't always right. For example, if analyzing software engineering data requires storing personal data in a database and this data is leaked, this may be a problem depending on if the data being analyzed is public or private in the first place. If a machine learning model inaccurately predicts the future work output of a software engineer, making them seem ineffective at their job for the near future, that would mean that the computation we have done is morally and ethically wrong.

I believe that we can still proceed with gathering, processing, and reasoning software engineering data for the benefit of the field. However, I think we must take caution with elements of it such as machine learning and GDPR issues with personal data. Errors in these regards can have devastating consequences. However, if we proceed with caution, I think the processing of this data can have a positive effect on the field of software engineering as we start to reason more about the software engineering process from not only a engineering standpoint, but a philosophical one too.

5 Conclusion

In this report, I have discussed methods of measuring software engineering, the platforms from which one could gather and perform calculations on software engineering data, the various types of computations that could be done on this data, and also the ethical and moral issues surrounding processing this data in the first place.

Software engineering is a complex field, usually with no easy answers. There are infinitely many possibilities, but also infinitely many things that could go wrong when reasoning about the software engineering data we gather. However, I believe that with the content in this report, one would have a good grasp of the things to be aware of while processing this data, and also the great impact that doing so can have on the software engineering community around the world.

6 References

1. Carsten Kolassa, Dirk Riehle, and Michel A. Salim. 2013. *The empirical commit frequency distribution of open source projects* [Online]. Available from: <https://doi.org/10.1145/2491055.2491073> [accessed 18th December 2021].
2. Renato R. Gonzalez. 1995. *A unified metric of software complexity: Measuring productivity, quality, and value* [Online]. Available from: [https://doi.org/10.1016/0164-1212\(94\)00126-8](https://doi.org/10.1016/0164-1212(94)00126-8) [accessed 20th December 2021].
3. Chris Bertrand. 2019. *Coding Concepts! Cyclomatic Complexity* [Online Image]. Available from: <https://dev.to/designpuddle/coding-concepts—cyclomatic-complexity-3blk> [accessed 21st December 2021].

4. Abram Hindle, Daniel M. German, and Ric Holt. 2008. *What do large commits tell us? a taxonomical study of large commits*. [Online]. Available At: <https://doi.org/10.1145/1370750.1370773> [accessed 22nd December 2021].

5. Austin Dressen. 2021. *Engineering Metric: Use Commit Size to identify risky code*. [Online Image]. Available At: <https://www.allstacks.com/blog/engineering-metric-commit-size> [accessed 23rd December 2021].

6. Juan Carlos González. 2019. *Teams Usage reports in the Teams Admin Center*. [Online Image]. Available At: <https://regarding365.com/teams-usage-reports-in-the-teams-admin-center-eae97b202258> [accessed 27th December 2021].

7. CloudHealth. 2018. *Teams Usage reports in the Teams Admin Center*. [Online Image]. Available At: <https://www.cloudhealthtech.com/blog/aws-vs-azure-vs-google> [accessed 29th December 2021].

8. Bruce G. Buchanan & Reid Q. Smith. 1988. *Fundamentals of Expert Systems*. [Online]. Available At: <https://stacks.stanford.edu/file/druid:gh880nv8408/gh880nv8408.pdf> [accessed 30th December 2021].

9. He, Qing & Wang, Yingxu & Far, Behrouz & Zhang, Shuangshuang. 2003. *A Web-based software engineering measurement expert system* [Online]. Available At: https://www.researchgate.net/publication/251424282_A_Web-based_software_engineering_measurement_expert_system [accessed 1st January 2022].

10. Tutorials Point. *Artificial Intelligence - Expert Systems* [Online Image]. Available At: https://www.tutorialspoint.com/artificial_intelligence/artificial_intelligence_expert_systems.htm [accessed 29th December 2021].

11. Tom Mitchell, McGraw Hill, 1997. *Machine Learning* [Online]. Available At: <http://www.cs.cmu.edu/afs/cs.cmu.edu/user/mitchell/ftp/mlbook.html> [accessed 1st January 2022].

12. Jean Helie, Ian Wright, Albert Ziegler. 2018. *Measuring software development productivity: a machine learning approach* [Online]. Available At: <https://codeql.github.com/publications/measuring-software-development.pdf> [accessed 1st January 2022].

13. Gotterbarn, Donald. 1991. *Ethical considerations in software engineering*. [Online]. Available At: <https://codeql.github.com/publications/measuring-software-development.pdf> [accessed 2nd January 2022].