Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

Measuring Software Engineering

Student Name: Alan Abraham

Student Number: 18322576

Introduction

To be able to discuss the idea of measuring software engineering, we need to know what it means exactly. From my understanding of the concept measuring software engineering seems to be a measure of how productive a software engineer is at his/her workplace. I will also discuss different metrics used for measurability along with the tools that one could use for this purpose.

Software Metrics

Software metrics are areas where measurement can be effectively applied to a specific software module or its specifications. Software metrics in simpler terms are just data taken from the software development lifecycle and the use of this data to measure the productivity of a software engineer. So how does one use this data to measure an engineer on the basis of productivity you may ask, this is quite a tough puzzle to solve as there is no concrete method to measure the efforts of an engineer. I will outline below the main software metrics that are measurable.

Most Reliable Software Metrics

Key software metrics that are used as a means to measure an engineer's productivity are below:

- Agile Process Metrics
- Code Churn
- Code coverage

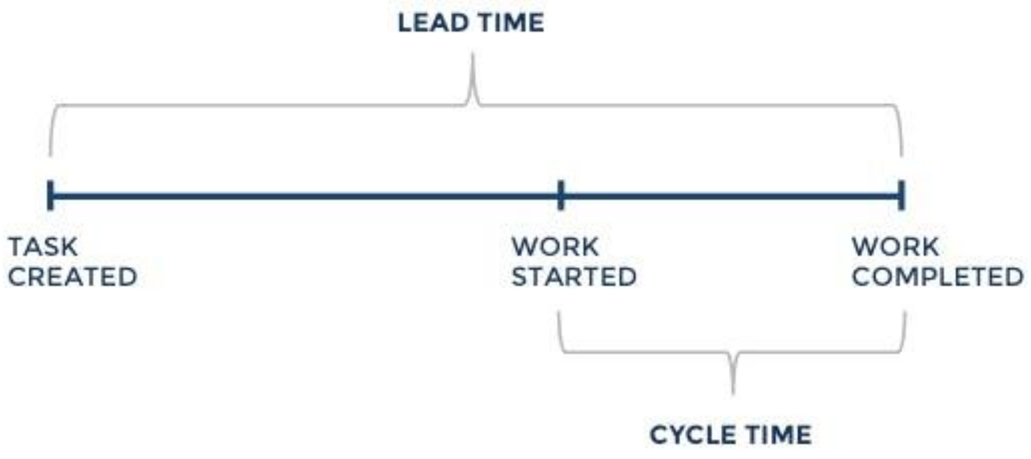
There are other metrics that are used as well but they are a not as reliable as the ones I have mentioned above nevertheless I will go into more detail about those metrics and their flaws below as well.

Agile Process Metrics

Agile is a software development approach discovering requirements and developing solutions through the collaborative effort of self-organizing and cross-functional teams and their customer/end user. The main metrics used in the Agile Process are Lead time, Cycle time and Team Velocity.

- **Lead time:** Lead time is the amount of time taken to go from an idea till the software is deployed. The lower the amount of Lead time taken the more productive the software engineer is. This is because the engineer has to be responsive to the end user and the faster, they complete the source code you could say the more productive they were.

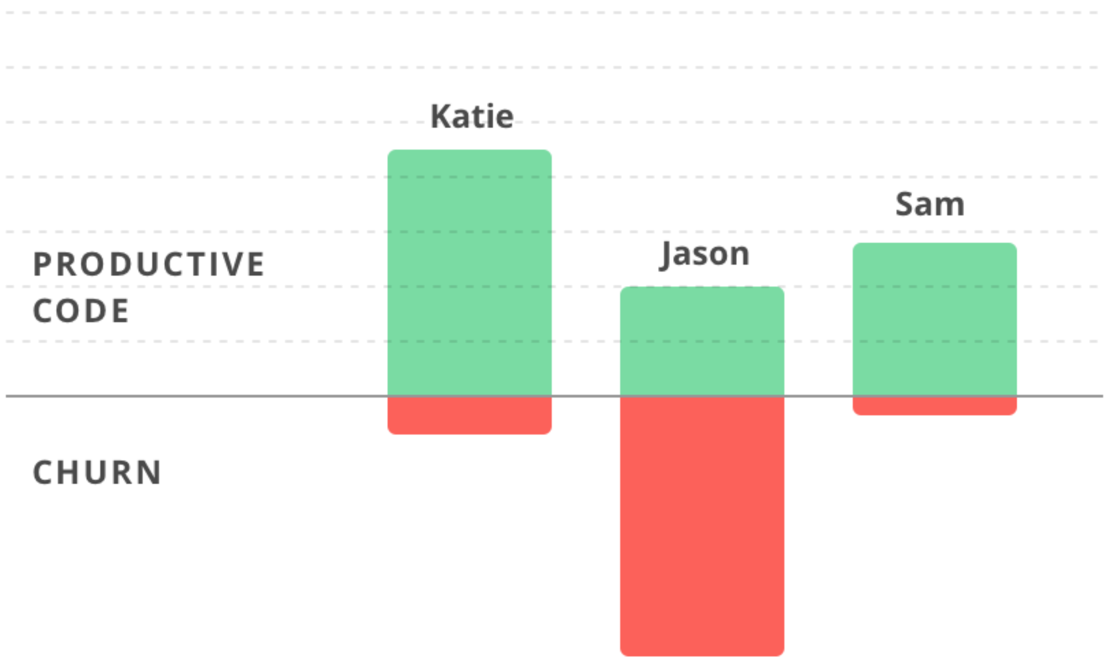
- **Cycle time:** Cycle time is the amount of time taken to make changes to the source code/software system and also the time taken for these changes to be deployed into production.



- **Team velocity:** Team velocity is basically a measure of a team's progress rate. Team velocity is measured by setting up multiple sprints for each of tasks that are part of the project. The Team velocity would be the rate at which a team completes each of these sprints. The faster a teams velocity is the more productive they are as a collection of software engineers.

Code Churn

The code churn is calculated by taking any code that was added, modified or deleted during a small period of time and is then divided by the total number of lines of code added. A code churn can be used to measure the productivity of a developer based on how fast they maybe solving a problem as the churn rate may indicate how well a developer is fairing with a problem. An example of code churn is shown in the image below.



Code Coverage

Code coverage is the amount of code that is covered by test cases out of the code bundle. The more code is tested the less bugs it will contain hence the more productive it is. Testing is very important for having less bugs in the code. Most popular programming languages have their own testing libraries for example in Java it is the Junit testing library and in Python it is the Unittest Library.

Unreliable Software Metrics

As I mentioned earlier in this report there are also some other software metrics that can be used even though they may be less reliable than the ones mentioned above, these metrics include:

- Hours worked
- Commits
- Lines of code
- Bugs fixed

Hours worked

Some people like to gauge how well a developer works based on the hours they have worked but productive developers will get more work done and solve more difficult problems in less time. The longer workers spend on problems the harder it maybe for them to complete the assigned work.

Commits

Another metric one could use to gauge productivity is commits, however commits can be very misleading as the number of commits for a developer can be a large amount but the quality of their code may not be good. Furthermore, a commit can be anything from changing a small part of documentation to adding large amounts of productive code, so commits cannot really be compared against each other to measure a developer's productivity.

Lines of code

Lines of code completed is another prospective way of measuring the productivity of a software engineer, however it is another unreliable metric. This is because developers can artificially increase the number of lines they produce for simple tasks to make it seem that they are more productive and also the fact that more lines of code doesn't mean the engineer solved the problem any better than any of his/her peers. Writing a lot of lines of code also makes the code less readable which is also a reason why lines of code are not a good measure of an engineer's productivity.

Bugs fixed

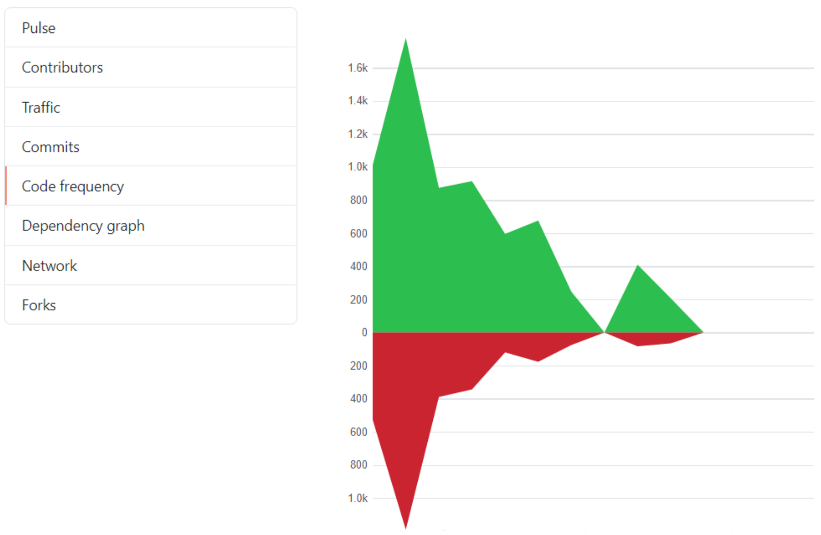
The final unreliable software metric that I will mention in this report is the number of bugs fixed. Much like the other unreliable metrics the number of bugs fixed is not a complete measure of an engineer's productivity as an engineer can artificially increase the number of bugs in his/her code and then solve those bugs. Another reason why number of bugs fixed does not measure a developers productivity is that each bug can be a different level of difficulty to fix, for example one developer could spend hours on one difficult bug while another fixes 20 in the meantime but that doesn't make the first developer any less productive.

Measuring Tools

- GitHub
- SonarQube
- Raygun
- Jira

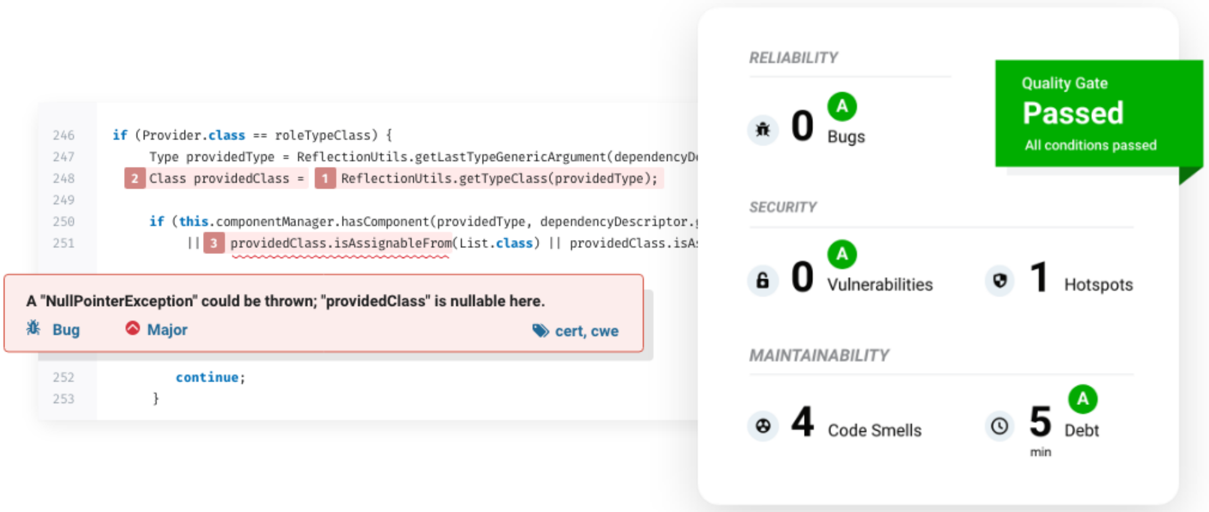
GitHub

GitHub is a code hosting platform used for version control and collaboration. It includes many features such as insights, pulse and commit history. GitHub also allows measures lines of code added and deleted. It allows users to measure pull and merge requests as well as issues. One can also use GitHub to maintain different branches of the same code bundle. Below is a screenshot of GitHub insights, where a graph of code frequency is shown.



SonarQube

SonarQube is an open source web platform used to provide quality checks in the user’s code. It supports around 25 languages and checks for bugs, duplications in code and other issues with the code bundle. It also provides analysis on the issues/bugs found in the code, which helps the software engineer improve their code. Below is an image that depicts how SonarQube works.



```
if (Provider.class == roleTypeClass) {
    Type providedType = ReflectionUtils.getLastTypeGenericArgument(dependencyD
  2 Class providedClass = 1 ReflectionUtils.getTypeClass(providedType);

    if (this.componentManager.hasComponent(providedType, dependencyDescriptor.
        || 3 providedClass.isAssignableFrom(List.class) || providedClass.isA
```

A "NullPointerException" could be thrown; "providedClass" is nullable here.

Bug Major cert, cwe

```
            continue;
    }
```

Raygun

Raygun is also bug tracking and network monitoring tool more suitable for corporations compared to the other tools. It also has crash reporting features that allow the developer to identify where the client experiences crashes or errors. Raygun also allows for workflow management and full stack application monitoring.

Jira

Jira is a well know software development tool, it is mainly used by teams of developers that use agile methodologies to optimise and improve workflows. Jira provides a lot of visual representation of data that could be extremely useful for the engineers that use it. These features include roadmaps, sprint reports, scrum boards and Kanban boards. These features allow for the lead time to decrease and the team velocity to increase.

Algorithmic approaches

Halstead’s Software Science/Metrics

These were some software metrics that were created by Maurice Howard Halstead in 1977 for his research on empirical science of software development. He realised that software metrics should reflect the implementation or expression of algorithms in different languages. The metrics he created are calculated from the code as follows:

- n1 = number of distinct operators in the program
- n2 = number of distinct operands in the program
- N1= total number of times an operator occurred
- N2 = total number of times an operand occurs

Using these values, we can calculate multiple measures:

- Length of the program: N = N1 + N2
- Volume of info: V = N log2(n1 + n2)
- Abstraction Level: L = (2*n2)/(n1*N2)
- Program effort: E = (n1 + N2*(N1+N2) *log2(n1+n2))/(2*n2)
- Time taken: T = E/18 seconds
- Number of bugs: B = B/3000

Where Volume V is the size of the program measured in bits, time taken is the amount of time taken to complete the program. The abstraction level is a metric that is used to measure how error prone a program is, the higher the abstraction level the less likely it will contain errors.

Halstead and his software metrics had a great impact in software measurement. He showed that the code complexity of a program is based on abstraction level and volume of the program.

Halstead’s approach seems to work for most cases however he received a lot criticism for his methodology and derivations of equations and other aspects of his research.

Artificial Intelligence as a software metric

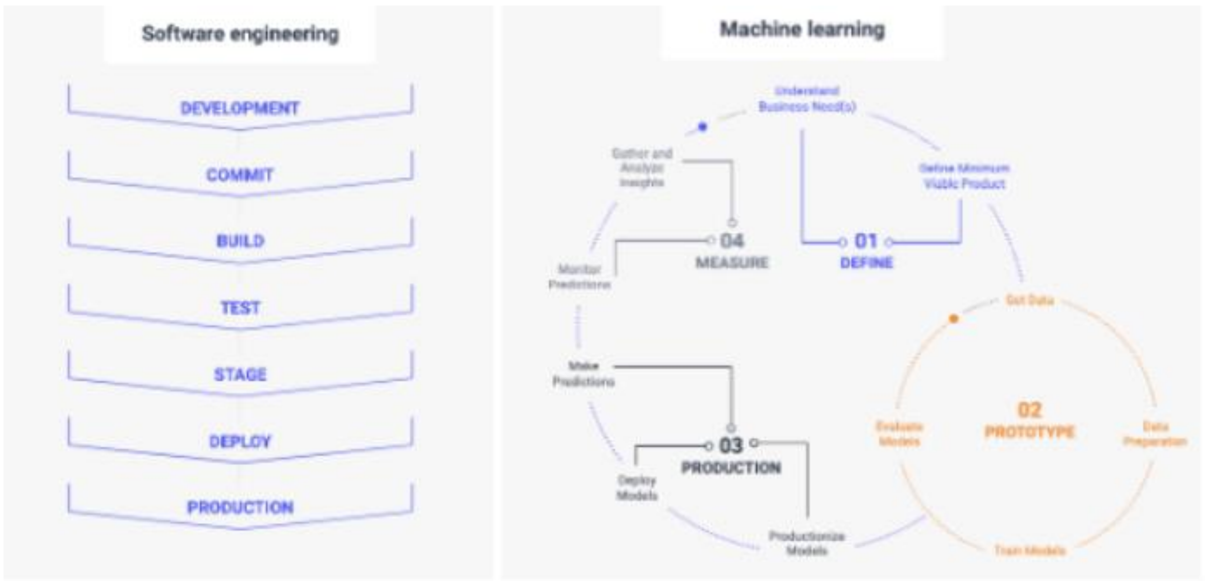
It is said that machine learning could be used to calculate, analyse and even predict a software engineer's productivity.

In supervised machine learning, the algorithm could learn form a training dataset and be used for predicting and possibly measuring the productivity of a software developer, however the dataset that it is trained on must be completely accurate and finite for this to be flawless. Some examples of supervised machine learning include linear regression and neural networks.

Unsupervised machine on the other hand could be more promising as it revolves around mapping input data to output variables. An example of this type of artificial intelligence is K-means clustering.

The final main type of machine learning is reinforcement machine learning. It allows developers to determine the ideal behaviour of a program within a specific context so that it can be accurate and as efficient as possible.

Maybe in the future we will have a generalised method of measuring software engineering using one of these types of machine learning.



Ethical Concerns

Measuring developers is something that is very important to most employers, however there are some ethical concerns associated with this. I will discuss these ethical concerns below.

Employee privacy

One of the main ethical concerns associated with measuring software engineers is employee privacy. The extent to which employers collect and monitor an employee’s information based on activities, communications and possibly even their private lives. Software engineers can be assessed in any form during work which could mean that employers could be collecting such data as mentioned above to measure the employee’s productivity. Obviously different corporations will have different privacy policies and regulations; however, employees are being monitored and their private data may be collected without their knowledge as part of a corporation’s regulations, which could be a breach of a workers right to privacy. What can be done to prevent such a breach of the workers privacy would be to enlighten them about how the company measures productivity and on what basis each engineer is measured, and the results should not be shared with an third party and stored securely.

Employee Autonomy

In software engineering, corporations are trying to embrace employee autonomy. This employee autonomy is one of the main reasons why each software engineer’s individual productivity becomes more important. Automation and delegation allow workers to increase the overall quality of their work and increase their productivity. Allowing for employee autonomy in the workplace allows each employee, their team and the whole corporation to achieve their benchmarks in terms of productivity.

Conclusion

In conclusion measuring software engineering is complex, however there are many ways to do so using the software metrics and tools mentioned earlier. These metrics and tools can be used to improve productivity rate and allows the software engineers to learn where they are making mistakes. One could even say that measuring a software engineer is the same as giving them feedback based on their productivity.

References/Bibliography

https://www.researchgate.net/profile/Jan_Sauermann2/publication/285356496_Network_Effects_on_Worker_Productivity/links/565d91c508ae4988a7bc7397.pdf

http://www.hitachi.com/rev/pdf/2015/r2015_08_116.pdf

Software Metrics

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

https://www.tutorialspoint.com/software_quality_management/software_quality_measurement_metrics.htm

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

https://pdfs.semanticscholar.org/4c43/5041f078e555737b49b04bd3ea6a92502036.pdf

Measuring Tools

https://docs.github.com/en/enterprise-server@2.19/insights/exploring-your-usage-of-github-enterprise/metrics-available-with-github-insights

https://www.sonarqube.org/

https://raygun.com/platform/crash-reporting

https://www.atlassian.com/software/jira

Algorithmic Approaches

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

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

Ethical Concerns

https://www.cnbc.com/2019/04/15/employee-privacy-is-at-stake-as-surveillance-tech-monitors-workers.html

https://ieeexplore.ieee.org/abstract/document/7436657

https://www.ciphr.com/advice/employee-autonomy/

https://ieeexplore.ieee.org/document/8170108