Claire Maria Gribbin - 16325614

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

SOFTWARE MEASUREMENT

Every company within the tech sphere wants to improve their software development process to use the minimum amount of resources possible to generate the maximum profit possible, they carefully estimate the cost, time, and product quality required. This is known as Software Measurement. To do this, software metrics are used.
"A software metric is a measure of software characteristics which are quantifiable or countable. Software metrics are important for many reasons, including measuring software performance, planning work items, measuring productivity, and many other uses.

Within the software development process, there are many metrics that are all related to each other. Software metrics are related to the four functions of management: Planning, Organization, Control, or Improvement." [1]

The crux of the matter is have a balance between quantity and quality, as to not sacrifice excellence in the pursuit of efficiency and allow developers to work at a pace that will not produce burnout.

In this report I will explore different ways in which software can be measured (software metrics), limitations of software measurement, applications of machine learning to aid efficient measurement and development and ethical issues within the scope of software development.

SOFTWARE METRICS - EXAMPLES

One variety of software metrics, is Agile Process Metrics:

These focus primarily on development teams, and do not measure the software instead they measure ways in which the development process itself can be improved and accelerated.

The Agile Process Methods I will be discussing are Lead Time/Cycle Time, Team Velocity, Code Churn and Happiness.

Lead Time:

Lead time defines how long it takes for ideas to be created and then delivered to production as software.

[1] https://dzone.com/articles/what-are-software-metrics-and-how-can-you-track-th

Cycle Time:

Cycle time is the length of time it takes to make changes to the software and implement that change in production ie. the time taken between two successful deliveries.

Lead/Cycle time is a direct measurement of the amount of work getting done per unit of time. Reducing this may include changes such as teams developing in parallel on various tasks.

" Calculating the cycle time provides information about the overall performance and allows for estimating the completion of future tasks. While the shorter cycle time illustrates better performance, the teams that deliver within a consistent cycle are valued the most."[2]

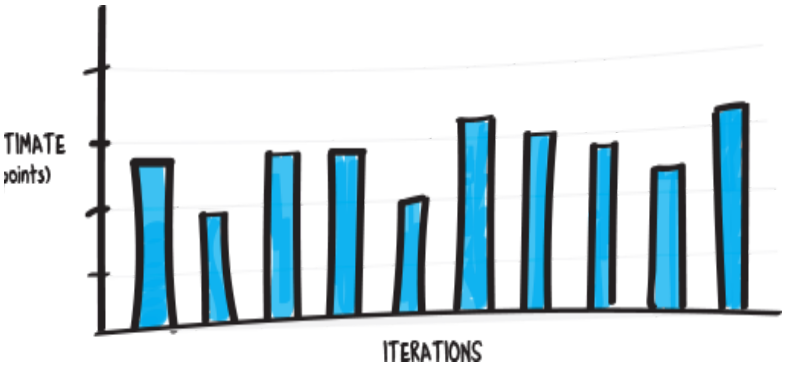
Team Velocity

Velocity is a planning tool to estimate how long a project should take. "The metric is calculated by reviewing work the team successfully completed during previous sprints; for example, if the team completed 10 stories during a two-week sprint and each story was worth 3 story points, then the team's velocity is 30 story points per sprint."[3]

Velocity is helpful for developers to avoid extending extra work into later iterations. Velocity tends to remain stable through the software development process and therefore is very helpful in taking accurate data samples and estimating the time needed for future projects accurately.

Changes in the team can easily be factored into velocity calculations, if 20% of the team are not available, or the team reduces in size by 20%, it is very simple to adjust the expected output of the team, you just reduce velocity by 20%. Therefore this is a very simple and useful tool for measuring the output of a team of software developers.

Therefore a velocity chart is a very simple bar chart, with point estimations on the y axis, and iterations on the x axis. Iterations can be any set amount of time. For example two weeks.



[2] https://whatis.techtarget.com/definition/Agile-velocity

[3] https://resources.collab.net/agile-101/agile-scrum-velocity

Code Churn

Code churn is when a developer rewrites their own code within a short space of time. This metric is specifically useful to avoid managers from thinking that the person who writes the most code is the best engineer. Code quality is equally important to code quantity, and taking code churn into account can show the most 'productive' members of the team - productive in the sense that they need less edits to their code, as many times people will find errors, incorrect behaviour and generally bad code after looking back over their work.
Image 1: From this graph it would appear that Jason has contributed the most to the project. This is without taking code churn into account.

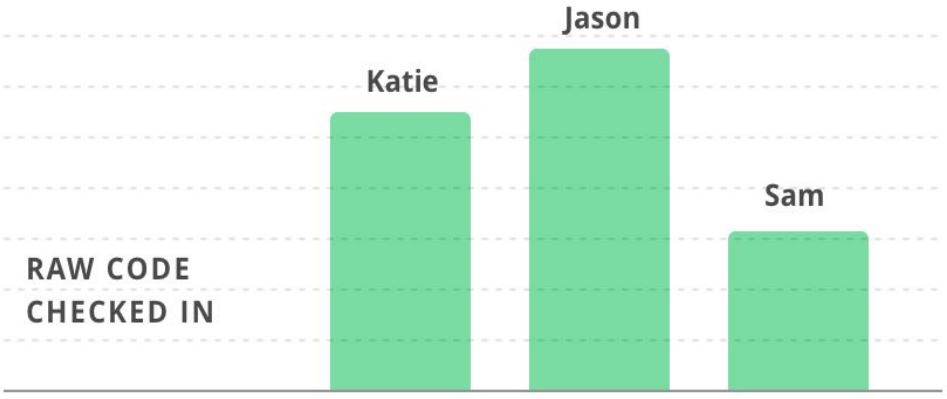
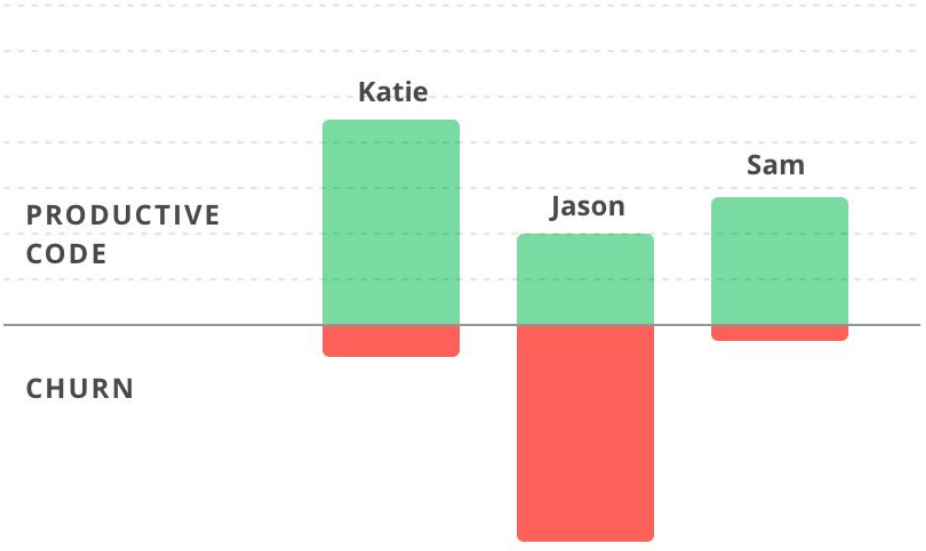


Image 2: Now with code churn taken into account you can see that in fact, Jason has spent a lot of his time rewriting code - this is not inherently a bad thing, but it shows a fairer balance of work between the team.



[4]

[4] https://blog.gitprime.com/why-code-churn-matters/

Happiness/Morale

Happiness may seem like a strange thing to measure when talking about software metrics, but I believe that it it vastly important that it should be considered in any work environment.

People who are unhappy with their work, or feel isolated are less productive as a whole. Trying to measure this, and act accordingly is currently rising in popularity among agile teams. Many companies vouch for this method, and recognize the need to care, in some way about their employees.

Desmart, an example of a company who uses this tool wrote: “Within our team it has helped a lot to diffuse tension or notice things that would have been overlooked otherwise. It is not uncommon to hear questions like “Why so black?!” when I mark a 3rd day in a row with a “so so” smiley or have the team demanding “Explain!” when somebody puts a red sad face on the board. It starts the discussion, so we can all together do something about the elephant sitting in the room.” [5]

NON AGILE METRICS - EXAMPLES

In this section I will be discussing some non-agile metrics, such as Lines of Code, Customer Satisfaction, Escaped Defects and Instruction Path Length.

Lines of Code

Also known as Software Lines of Code (SLoC): This is a software metric that determines the size of the program by counting the lines of code in the source code of the program.

This is mostly used for programmers to determine how much work is left to be done, and the effort that it will take. There are two major types of SLOC measures: physical SLOC (LOC) and logical SLOC (LLOC).

Physical SLOC: This is a count of the lines of code in the source code, including comments and blank lines as long as the blank lines make up less than 25% of a code segment.[6]

Logical LOC: This measures the amount of ‘statements’ made within the code. How this is measured is determined by the language being used. One simple logical LOC measure for C-like programming languages is the number of statement-terminating semicolons. [7]

[5] https://desmart.com/blog/how-measuring-happiness-helped-us-build-a-better-team
[6] http://www.projectcodemeter.com/cost_estimation/help/GL_sloc.htm
[7] http://www.projectcodemeter.com/cost_estimation/help/GL_sloc.htm

Customer Satisfaction

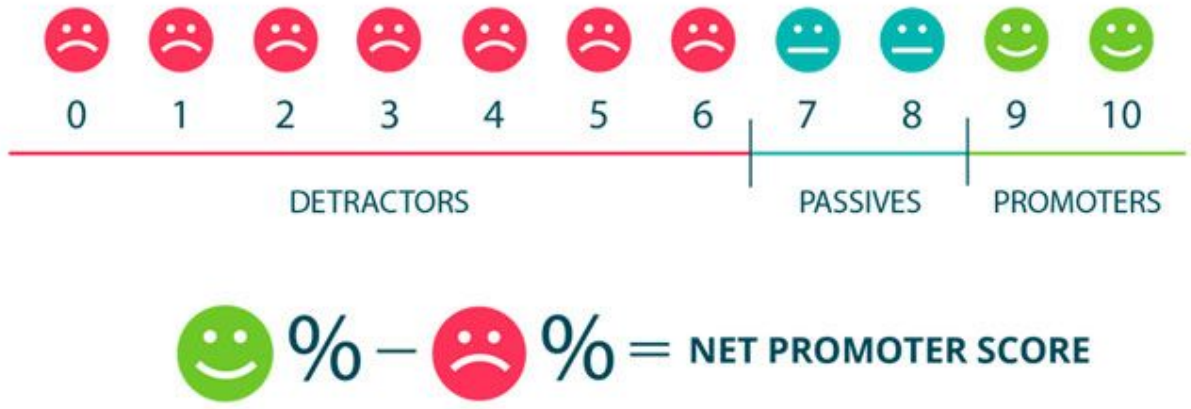
Part of code quality is how happy shareholders and customers are with the software. There are multiple ways of measuring this. Customer satisfaction is a measure of how products and services supplied by a company meet or surpass customer expectations. [8]

The simplest way of measuring Customer Satisfaction is a survey. Typically sent out via email. Asking simply “are you happy with the product?” may not be enough, typically you would ask them to rate their experience out of a given number, and ask about various parts of the software and the users experience with each.

A widely used metric in measuring Customer Satisfaction is the Net Promoter Method. It asks if the user would recommend this product.
Responses in the 9-10 range are considered promoters, 6-8 are considered passive and 0-6 are considered detractors.

The formula used to calculate the final score is: (# of Promoters — # of Detractors) / (Total Promoters + Neutral + Detractors) x 100. [9]

[10]



[8] https://tcagley.wordpress.com/2016/07/21/customer-satisfaction-metrics-and-quality/
[9] https://tcagley.wordpress.com/2016/07/21/customer-satisfaction-metrics-and-quality/

[10] https://www.sealights.io/software-development-metrics/top-5-software-metrics-to-manage-development-projects-effectively/

Escaped Defects

This measures the amount of bugs released in production, and found afterwards. This is a good metric to measure code quality. If this number rises it is clear that there is something wrong with the quality process and the testing of the code.

Instruction Path Length

Instruction Path Length measures the amount of machine code instructions needed to run a section of code. The complete instruction path length measures the amount of time needed to run the software on a specific hardware. This is mostly a relative measure of speed considering hardware can greatly affect the runtime of programs.

SOFTWARE METRICS - LIMITATIONS

Lead/Cycle Time:

Problems may arise when the issues facing the team come from mismanagement of time or poor communication between team members. Individual members of the team may be excellent developers but unless a team can cooperate problems will always arise. The issue is measuring lead/cycle time alone does not cover why slowdown may occur, or why certain teams underdeliver.

Velocity:

Velocity is defined with respect to units of value (user stories) rather than with respect to units of effort (tasks). There is no meaningful comparison of velocity "between" different teams, since such teams may have different approaches to measuring velocity, and different teams may have different levels of difficulty in their tasks. [11]

Code Churn:

There is no real way to compare each team against the average since code churn will be different for different teams. It should only be used as a measure of efficiency of that particular team or department. Code churn will also not always be consistent over the development cycle, and changes in churn should be taken into account when evaluating the team.

---

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

Happiness:

Many people have different measures of happiness and hold different values. Therefore there is no real way to measure happiness over a group of people as simply asking the question, “are you happy?” is very open to interpretation. Plus, many people are simply ‘OK’ the vast majority of the time and this isn’t a major problem.

Happiness is very vague, to be happy with your company is one thing, but one can be happy with the company they work for, but hate their team. I believe to even get any kind of usable data this question, or questions must be asked anonymously. This is due to the fact management may not like the honest answers, or people may be afraid to lose their jobs and answer with full positivity even if they don’t like aspects of their job.

“It’s possible to have a completely happy team that is not performing at all. You can have a happy team that produce sloppy, buggy code and does not complete sprints on time. In fact, the most happy team members are probably those hanging out in the XBox Game Room. Asking if someone is happy does not give you _any _meaningful insights into the team’s well-being. Happiness, after all, is a very individual state of mind.” [12]

Lines of Code

Implementation of a specific logic differs based on the level of experience of the developer. An experienced developer may write the same function in fewer lines of code than a less experienced developer, even though they are using the same language. This is not a very good means of measuring skill, or dedication to the project.

Different languages will also yield different amounts of code, for example a Java program will be longer than a Scala program.

Also, measuring performance off of lines of code written will incentivise the programmers to write unnecessarily verbose code, meaning that quality, and speed can suffer as a result. A lot of badly written code is not better than less code, but equal functionality and better, quicker code.

APPLICATIONS OF MACHINE LEARNING IN SOFTWARE MEASUREMENT

“Machine learning (ML) is a field of artificial intelligence that uses statistical techniques to give computer systems the ability to "learn" (e.g., progressively improve performance on a specific task) from data, without being explicitly programmed”[13]

---

[12] https://medium.com/the-liberators/agile-teams-dont-use-happiness-metrics-measure-team-morale-3050b339d8af

[13] https://en.wikipedia.org/wiki/Machine_learning

Machine learning allows a system to grow and learn from itself without the need to be programmed continuously.

Learning Replacing Code

"Machine learning is already making code more efficient: Google's Jeff Dean has reported that 500 lines of TensorFlow code has replaced 500,000 lines of code in Google Translate. Although lines of code is a questionable metric, a thousand-fold reduction is huge: both in programming effort and in the volume of code that has to be maintained."[14]

Neural Networks are the basis of this, and studies show they can be used to create new code based of existing modules. At the very least, it could optimise small amounts of the code and help developers write better, cleaner code, quicker.

Software Testing

Traditional testing methods rely on humans to analyze data, and draw conclusions. Human error is a large problem in the industry and could potentially be eliminated, or reduced by machine learning. The time needed to perform a software test and find problems is also reduced, while the amount of data that needs to be handled can still increase without any strain on the testing team. [15]

Predictive Analysis

Predicting customer needs based on past data, using machine learning algorithms could help companies greatly by identifying trends in the marked and foreseeing problems before they ever happen.

Bug Prediction

It is also possible to use a ML algorithm to detect where a bug may occur and allow developers to identify problems in the code easily, or before they happen. If this were to be used, it would easily speed up the time it takes to produce products.

ETHICS

Ethics is the study of whether or not something is 'good' or 'bad'. In the Tech Industry this means that it needs to be decided what the can or cannot do. For example, many consider selling user data to be unethical, but many companies do this.

I will be discussing privacy

---

[14] https://www.oreilly.com/ideas/what-machine-learning-means-for-software-development

[15] https://towardsdatascience.com/how-machine-learning-and-ai-bring-a-new-dimension-to-software-testing-7b2b6ea67b61

Privacy

Data protection is the first thing that comes to mind when ethics is brought up regarding software engineering.

It is ethical to protect your users' data. This means choosing tools, platforms, APIs, etc which allow true deletion.

Passwords should also be stored using encryption as unencrypted passwords lead to security issues and potential breeches. You should implement password resetting rather than recovery. Store passwords in MD5 or other unrecoverable forms.

Data is often sold to advertising companies to predict what a user will buy and target advertisements at them, many websites offer data protection but it often involves reading through text, or unchecking boxes which many people do not do. Some companies also change their data protection policies without notifying their users.

Location Privacy is also very important, location based services can cause issues, certain apps that track a user's location can be used to direct advertising based off the location of the user. Location tracking can also be used by other people for nefarious reasons. (ie. stalking).

It is important to prompt the user to evaluate settings regarding location services and allow the user to turn location on and off depending on whether they want to use it.

Accuracy/stability a system that delivers consistent results is very important. Software working improperly can have disastrous effects on the user base.

For example, if a man, lets call him Mike wants to make a payment that is urgently needed to send his daughter to school but the payment is rejected due to insufficient funds. He needs to make the payment by the end of the day. The bank confirms that his account had plenty of funds the day before, but cannot tell him why the funds are gone now or why the payment was rejected. They tell him there must be some 'software glitch' involved and that they will open an investigation, but that it will take weeks to resolve. They will only restore the funds in his account once the investigation is completed and the cause found. [16]

This is an example of a small glitch having disastrous effects on an individual's life, as his daughter must now figure out what to do, and see if she can attend college.

It is very important during the development cycle to test all possible states of the program and ensure to the best of your ability, that the program has very few or no problems.

[16] https://www.scu.edu/media/ethics-center/technology-ethics/Students.pdf

BIBLIOGRAPHY

1. https://dzone.com/articles/what-are-software-metrics-and-how-can-you-track-th
2. https://whatis.techtarget.com/definition/Agile-velocity
3. https://resources.collab.net/agile-101/agile-scrum-velocity
4. https://blog.gitprime.com/why-code-churn-matters/
5. https://desmart.com/blog/how-measuring-happiness-helped-us-build-a-better-team
6. http://www.projectcodemeter.com/cost_estimation/help/GL_sloc.htm
7. http://www.projectcodemeter.com/cost_estimation/help/GL_sloc.htm
8. https://tcagley.wordpress.com/2016/07/21/customer-satisfaction-metrics-and-quality/
9. https://tcagley.wordpress.com/2016/07/21/customer-satisfaction-metrics-and-quality/
10. https://www.sealights.io/software-development-metrics/top-5-software-metrics-to-manage-development-projects-effectively
11. https://www.agilealliance.org/glossary/velocity/#q=~(infinite~false~filters~(postType~(~'page~'post~'aa_book~'aa_event_session~'aa_experience_report~'aa_glossary~'aa_research_paper~'aa_video)~tags~(~'velocity))~searchTerm~'~sort~false~sortDirection~'asc~page~1)
12. https://medium.com/the-liberators/agile-teams-dont-use-happiness-metrics-measure-team-morale-3050b339d8af
13. https://en.wikipedia.org/wiki/Machine_learning
14. https://www.oreilly.com/ideas/what-machine-learning-means-for-software-development
15. https://towardsdatascience.com/how-machine-learning-and-ai-bring-a-new-dimension-to-software-testing-7b2b6ea67b61
16. https://www.scu.edu/media/ethics-center/technology-ethics/Students.pdf