Measuring Engineering

Measurable Factors

There is a huge amount of different data that can be gathered on an individual software engineer. For this report I have broken it down into three categories: Code - data on the type of work the engineer is doing, Environmental - data on the work environment, and Personal - data specific to the individual being measured. Some of this data provides information on the performance of the engineer while some of it is more suitable for examining factors that could be influencing performance.

Code

The data gathered here could be gathered through a version control system such as Git.

Code Quantity

This is perhaps the most simple way to measure the amount of work done by an individual software engineer: a count of the number of lines of code written.

The concept of source lines of code or SLOC[1] involves measuring the number of lines of code written by an engineer. This metric however has several issues: simply measuring performance by the volume of code produced does not take into account the format of the code written, for example the following snippets are identical in their effect:

```
for (i = 0, i < some_num, i++)
      do_a_thing()
```

```
i = 0
while (i < some_ num){
      do_a_thing()
      i++
}
```

Simply measuring the volume of code produced by an engineer could lead to individuals padding their performance statistics by writing very verbose code.

Another way of measuring the volume of code produced is by measuring the logical lines of code or LLOC written[2]. This is a measure of the number of statements within the code, disregarding formatting. In the case of the previous example this could break down into:

[1] https://en.wikipedia.org/wiki/Source_lines_of_code
[2] http://www.projectcodemeter.com/cost_estimation/help/GL_lloc.htm

```
for (i = 0, i < some_num, i++) # 4 LLOCs
      do_a_thing()             # 1 LLOC



# Total LLOC : 5
```

```
i = 0                          # 1 LLOC
while (i < some_ num){         # 2 LLOC
      do_a_thing()             # 1 LLOC
      i++                      # 1 LLOC
}
# Total LLOC : 5
```

As this demonstrates, LLOC can overcome some of the issues with SLOC. However this metric is less transparent than simply measuring lines of code and it can still be cheated in various ways. It is also far more difficult to implement a fair way to count logical lines of code than physical lines.

Bugs per Line[3]

One method used for analysing the quality of code is the measurement of how many bugs or errors exist in an average line of code. This is usually measured in bugs per 1000 lines of code and provides some insight into the quality of the software being written as well as the skill of the engineer writing it and their familiarity with the tools they are using.

Consistency of Code

This is a measurement of an engineer's performance over time.

One way to measure the consistency of work is to consider the quantity of code produced with respect to time. Ie evaluate SLOC or LLOC as a function of time. This information could be very useful in gauging a software engineer's performance but also suffers from several pitfalls eg. it does not take into account the difficulty of the work being done.

Another metric that can be used is 'churn rate' [4], this is the percentage of a developer's code that is an edit to their recent work[5] ie. how much of their code they are having to rewrite. A high churn rate can indicate low quality code being written whereas a low churn rate can indicate that the code written was of a higher quality. Churn rate can also be an indicator of how clear a project goal is as constantly changing objectives will lead to a higher churn rate.

Commit frequency could also be used to gain insight into the software development process. The number of commits in a day as well as the pattern of commits over time could be used to analyse performance. For example does an engineer commit very few times a day or does the amount of work they do reduce towards the end of the week.

---

[3] https://www.mayerdan.com/ruby/2012/11/11/bugs-per-line-of-code-ratio
[4] https://scottbarstow.com/churn-in-software-development/
[5] https://stackify.com/measuring-software-development-productivity/

Code Review

Pull requests or code reviews can also provide useful insight into work being done by a software engineering team. Since pull requests allow other members of a team to comment of code written by others it can be an opportunity to gain insight into systemic issues within the team[4] and also to find ways in which to improve the efficiency of the team.

During code reviews the readability of the code as well as how clearly it is commented can also be measured. While this may be quite subjective it could help improve processes by guiding the change of coding standards.

The way in which feedback is delivered to the original developer during a pull request can also be indicative of how well a team works together. An example would be determining whether the feedback is constructive or just critical.

Bug Fixes

Fixing bugs can often be a more involved process than writing the original code. The process requires an understanding of the problem code and how it interacts with other parts of the software. Some bugs can be very challenging to understand, let alone fix and this can lead to them being very slow to solve. During this time the quantity of code a developer produces could be quite low which makes their performance harder to measure as it is dependent on the difficulty of the problem which is again, hard to measure.

Environmental

The working environment can also be measured to assess its impact on software engineer performance.

Office Climate

Variables such office temperature and light levels or sources can have an impact on an individual's performance. Office temperature has a measurable impact on performance with temperatures too high or too low having a detrimental impact on productivity of people working in the environment[6]. The type and quality of lighting in the office can also can also have a significant effect on individual performance[7]. If the end goal of measuring performance is to improve it then these should also be considered.

---

[6] https://indoor.lbl.gov/sites/all/files/lbnl-60946.pdf

[7] https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4031400/

Communication With Coworkers

Conversations between software engineers in the workplace could be analysed to gain insights from conversation tone or length[8]. Furthermore conversation content could be analysed to infer how focussed developers are on their work. This information also has the potential to provide some insight into team dynamics by providing detailed information on how colleagues treat each other in the workplace.

Personal

Health

Keeping track of various data relating to a software engineer's health could allow for a more detailed analysis of their performance and also improve health and productivity by providing the ability to both prevent and diagnose health problems that are currently affecting performance or could do so in the future.

Measuring posture while working could provide information on how focussed an engineer is and also provide recommendations to improve comfort, performance and long term health[9].

By combining this information with other data such as heart rate, breathing rate and blood oxygen levels, stress could be measured[10] and links established between stress and performance.

Hobbies

What an engineer is doing in their free time could affect their performance at work. By gathering this information, a clear relationship could be established between engineering performance and activities outside of work. For example exercise[11] has a proven link to job performance. This could allow for improving engineer health and performance by providing insights into physical and mental health[12].

Computational Platforms

There are a large number of computational platforms that aid in the measurement and analysis of software engineers and engineering processes. I have selected a few and split them into two groups: Cloud services - services offered by companies

[8] https://en.wikipedia.org/wiki/Humanyze

[9] https://ijooes.fe.up.pt/article/view/2184-0954_002.002_0005/89

[10] https://assets.firstbeat.com/firstbeat/uploads/2015/10/How-to-Analyze-Stress-from-Heart-Rate-Variability.pdf

[11] https://www.omicsonline.org/open-access/the-relationship-between-physical-exercise-and-job-performance-themediating-effects-of-subjective-health-and-good-mood-2223-5833-1000269.pdf

[12] https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1470658/

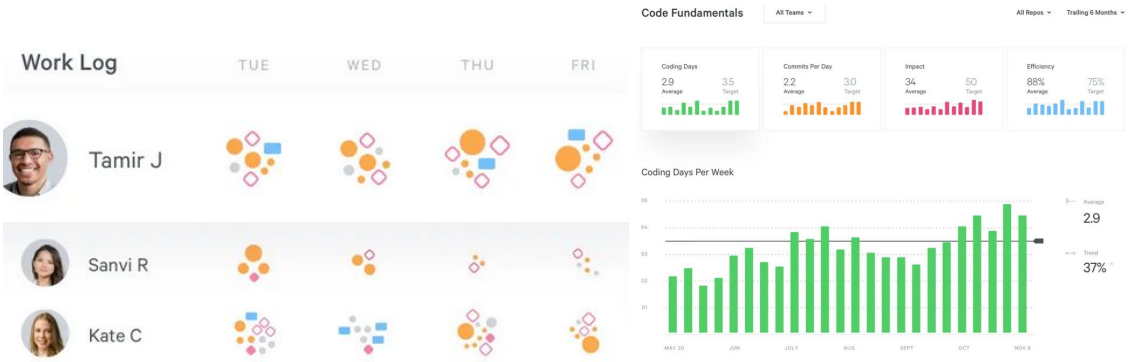
that perform measurement and analysis, and Frameworks - software providing general functionality that can be incorporated into existing code and built on.

Cloud services

GitPrime[13]

GitPrime is a cloud-based service that analyses data gathered from the version control system being used by a company or individual (eg GitHub, BitBucket, SVN). They claim to analyse a large number of variables surrounding the code being produced, as well as pull requests and the type of interactions between engineers in pull requests. By doing this they claim to be able to provide insights into the performance of a specific engineer or a team over time. They provide data on velocity, the type of work done, churn, etc. Another company providing a similar service is Waydev[14].

Examples of data visualisations from GitPrime:



Velocity 2.0[15]

Velocity is a product by the company Code Climate. Similarly to GitLab, they focus on analysing data from a repository and drawing information from this. However unlike GitPrime, whose primary focus is on analysing an individual developer, Velocity is aimed at finding process constraints in departments. They also focus on visualising raw engineering data so that it can be understood by people who may not be software engineers or have experience with software engineering. Using this information they aim to provide a clear view of progress over time.

Frameworks

Hackystat[16]

Hackystat is an open source framework for collection, analysis, visualization, interpretation, annotation, and dissemination of software development process and product data. Hackystat allows users to gather data by attaching software ‘sensors’ to their development tools. These then feed this data back to server set up by the

---

[13] https://www.gitprime.com/
[14] https://waydev.co/
[15] https://codeclimate.com/
[16] http://csdl.ics.hawaii.edu/Research/Hackystat/

developer or company. This server then provides a suite of features to visualise and analyse this data. Since the framework is designed to gather data from development tools, it also allows the engineer to analyse work other than software such as documentation. Since Hackystat is a framework designed for the user to set up, it does not place time limits on data analysis, unlike the cloud services previously mentioned which have a limit of 3 months on historical data.

Algorithmic Approaches

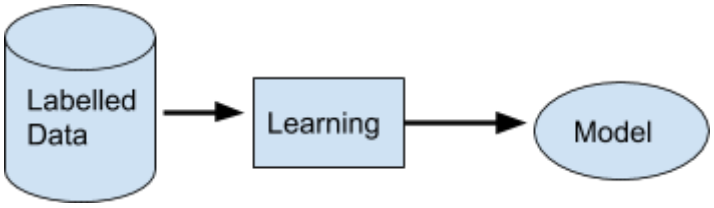
AI / ML analysis[17][18]

AI or machine learning are increasingly being used to analyse large amounts of data. These methods have proven to be very useful in tackling large problems such as translation[19], image recognition[20] and self-driving vehicles.There are three main methods of machine learning 'training' methods which will be outlined below.

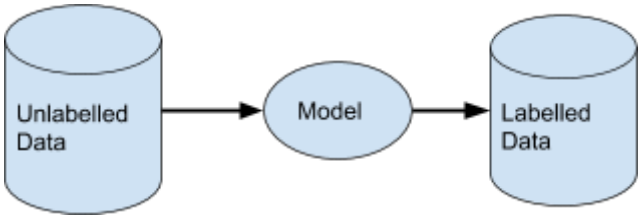
Supervised learning[21]

This is the term used to describe a method of training an algorithm by providing it with a large labelled dataset. The algorithm learns from this and after it has learned from this data, the resulting model uses it when asked to analyse other data. For example: An image recognition algorithm could be given a large number of pictures of cats, it 'learns' from this data and the resulting model, when given a different image ideally will be able to accurately determine if the picture contains a cat.

Training using supervised learning:



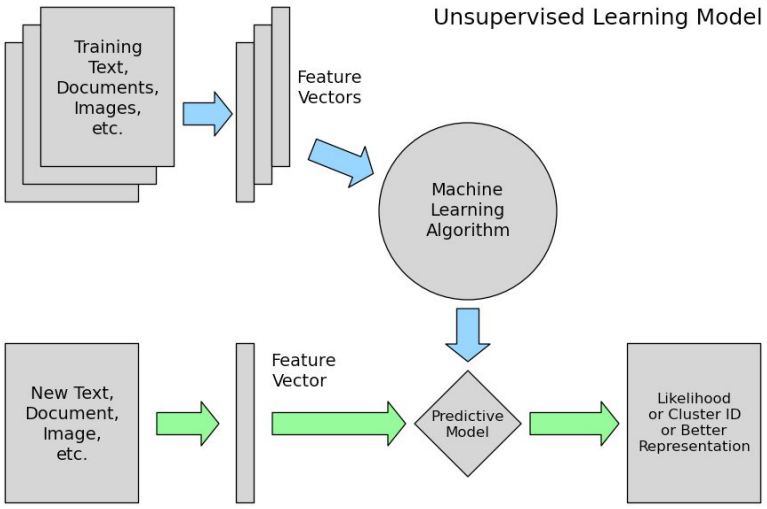
Testing the model:



[17] https://www.dummies.com/programming/big-data/data-science/3-types-machine-learning/
[18] https://www.educba.com/types-of-machine-learning/
[19] https://en.wikipedia.org/wiki/Google_Neural_Machine_Translation
[20] https://cloud.google.com/vision/
[21] https://en.wikipedia.org/wiki/Supervised_learning

Unsupervised learning[22]

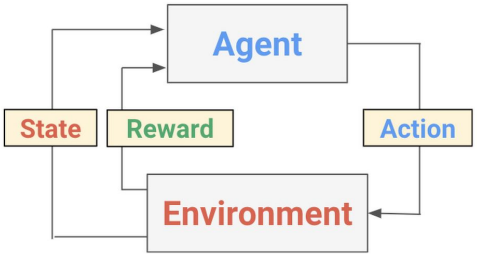
This method involves providing the algorithm with a large amount of unlabelled data and allowing it to discover correlating factors on its own. An example of this kind of trained system can be found on some online shopping recommendations, given a large amount of data on what people bought, the algorithm will make assumptions on what products to recommend. Recommendations are based on what other people previously purchased with a similar product.



Reinforcement learning[23]

This is similar to unsupervised learning in the sense that the algorithm is provided with unlabelled data. However this data is accompanied by positive or negative feedback depending on the output of the algorithm. This is similar to learning by trial and error. This type of learning is often used when teaching algorithms to play games. This is often easier as the game will usually have a clear outcome. An example of this can be seen with Google DeepMind learning to play Atari Breakout[24] . An advantage of this is that the algorithm can learn different techniques to solve a problem without any previous input, it only looks for the best way to solve the problem.

Reinforcement learning model



.

[22] https://en.wikipedia.org/wiki/Unsupervised_learning
[23] https://en.wikipedia.org/wiki/Reinforcement_learning
[24] https://youtu.be/V1eYniJ0Rnk

Statistical analysis

The data could also be analysed by using tools such as SAS, SPSS, R, MatLab and Excel. These would require a large amount of work to be done in order to use these tools for proper statistical analysis.

Ethical Concerns

There are a huge range of ethical concerns that need to be taken into consideration when approaching this topic. For the purpose of this report I have split them into 3 broad topics: concerns about the data being gathered, concerns about the analysis of the data and concerns over the interpretation of the data being measured.

Concerns about gathering data[25]

Gathering data based on code is of little ethical concern as this data is already being made available by the engineer. Since this data is the work that is provided by a developer, it is ethically sound to analyse, however it could be manipulated by a software engineer in order to appear more productive, as can be seen in the previous example relating to SLOC.

Personal and health data gathering is, I believe highly unethical. Constantly gathering data such as heart rate or blood pressure provides a huge amount of insight into a person's life, something I believe an employer should not have access to. The collection of this data could allow employers to control aspects of an individual's life outside of their job, for example if a smartwatch or fitbit shows that an employee has not been exercising recently, their employer could be informed of this. Leading to the question: should an employer have the right to set expectations on what is appropriate in terms of exercise or sleep?
Another example of this would be the recording of employee voice data, recording conversations between individuals. This type of monitoring is very invasive and could put a lot of pressure on employees to say the 'right thing'. It could also give an employer the opportunity to police people's thoughts and opinions.

After this data has been gathered it must be stored securely given its sensitive nature. There have been many large scale data breaches in recent years such as the case of Equifax[26] where the social security numbers, birth dates, addresses and even drivers licences of 143 million Americans were stolen. In August 2013, data on every Yahoo customer was stolen[27], this amounted to over 3 billion accounts[28]. With the scale of these corporations and the sensitivity of the data they had access to, it could be assumed they would take steps to properly safeguard this information.

[25] https://www.hrmagazine.co.uk/article-details/the-ethics-of-gathering-employee-data
[26] https://www.consumer.ftc.gov/blog/2017/09/equifax-data-breach-what-do
[27] https://www.malwarebytes.com/data-breach/
[28] https://haveibeenpwned.com/

However if these huge companies have proven themselves vulnerable to attacks, it can be reasoned that if large amounts of software companies were collecting and storing personal data on their employees such as health data, there is a significant chance it would be stolen.
This data doesn't need to have been stolen by someone from outside the company, several people inside a company could have access to this sensitive information. It would be very important to carefully select who should have access to this information.
Data permanence is another concern, many companies[29] store many backups of their data, this is essential to help prevent data loss. However if a company had a large amount of personal data regarding an employee and this person then leaves the company, should they be allowed to retain the data?

Concerns about analysing data

This really boils down to: is trusting a machine to measure a person okay? There are many issues with techniques used to analyse this data, many of them surrounding machine learning. One major issue with using this type of algorithm is that it is not guaranteed to produce the correct result. For example, researchers were able to trick an image recognition program into identifying a cat as a guacamole[30], another, more serious example would be when a self driving test vehicle failed to identify a person crossing the road and fatally injured the pedestrian[31]. This was despite the vehicle's sensors having detected the pedestrian in advance. This highlights the risk of allowing a computer to make important decisions.
Another concern with using these technologies is that we have very little ability to see the internal workings of such an algorithm. This can make problems very hard to fix, for example Google's image recognition algorithm previously "tagged pictures of black people as 'gorillas' "[32]. The company initially solved this by removing the tag 'gorilla', it took over two years for a proper fix to be implemented. This demonstrates a key issue with machine learning, a lack of control over what the software does.
These algorithms have also been shown to exhibit bias[33][34], if an algorithm was trained primarily on data (voice data of conversations between colleagues) containing the voices of men (who currently make up a majority of software engineers), it could theoretically develop a bias towards males employees and score female employees lower. This difference may only be very subtle but could

[29] https://www.google.com/about/datacenters/data-security/
[30] https://arxiv.org/abs/1707.07397
[31] https://www.theguardian.com/technology/2018/mar/22/video-released-of-uber-self-driving-crash-that-killed-woman-in-arizona
[32] https://www.theguardian.com/technology/2018/jan/12/google-racism-ban-gorilla-black-people
[33] https://www.theguardian.com/technology/2017/apr/13/ai-programs-exhibit-racist-and-sexist-biases-research-reveals
[34] https://medium.com/coinmonks/ai-doesnt-have-to-be-conscious-to-be-harmful-385d143bd311

nonetheless have a significant impact on the perceived skill or value of the software engineer.
Given these concerns it reasonable to question whether the results produced by the analysis of this data are an accurate measure of an individual's performance.

Concerns about the interpretation of measured data

Another concern when dealing with this information is that the person interpreting the resulting data may not understand the context of it. It could also be presented in a misleading way[35] or simply misunderstood by someone who has very little knowledge of the subject matter. This misinterpretation could cost someone their job as the value of their work may not be fully understood.

[35] https://en.wikipedia.org/wiki/Misuse_of_statistics