

Software Engineering: Measurement & Assessment

Ruairí O'Connor - 17330931

Introduction

Measuring and evaluating software engineering processes has its major benefits and drawbacks. When used correctly it can give a clear indication of how well a project/developer is functioning. It can indicate the efficiency and skill that goes into the development of software engineering processes as well as highlighting places in which these processes can be optimised. When used incorrectly, it can be seen as an invasion of privacy and may not accurately represent the project/developer.

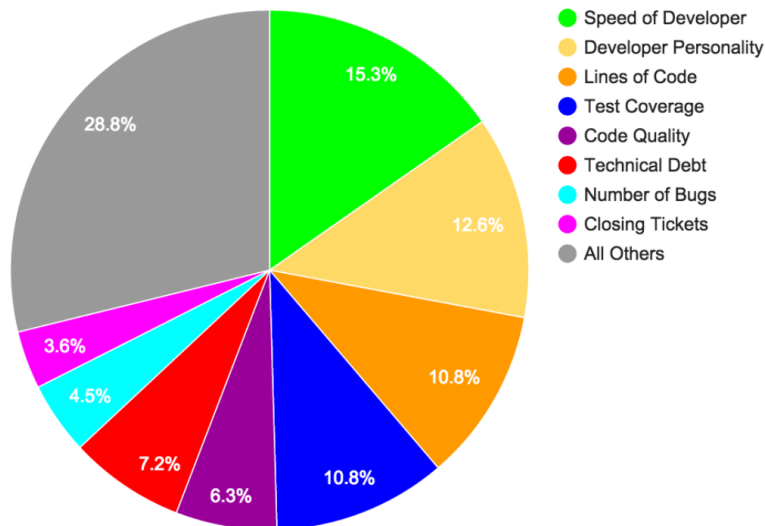
In this report, I hope to break down this process into four main categories:

1. The ways in which the software engineering process can be measured and assessed in terms of measurable data.
2. Give a brief overview of the computational platforms available to perform this work
3. The algorithmic approaches available.
4. The ethics surrounding this kind of analytics.

I will also provide a conclusion of the paper where I will give a brief overview of the topics discussed as well as my own opinion on each.

Measurable Data

The first step in measuring and assessing software engineering processes is defining by what data can we measure these processes by and how we obtain this data. In the software engineering industry, there is many metrics by which processes can be assessed and even more methods for obtaining them.



This pie chart is taken from an article exploring developer performance metrics (York, 2015). It is an accumulation of the performance metrics that a group of 300 software developers thought were the most important.

1. **Speed of Developer:** This metric is pretty straight forward and does not have to just apply to developers, it is how fast a project can be completed. It includes lead time and cycle time. It may be hard to asses as each project will have its own level of complexity and length.
2. **Developer Personality:** This metric may be very specific but is still an important on it includes how well a Developer works with other developers and their communication skills when it comes to presenting the project.
3. **Lines of code:** This metric is an obvious one but can also be misleading or not accurate as again every project is different, and it does not consider code quality.
4. **Test Coverage:** Testing code is essential when producing a project and you should test every line of code to ensure it works correctly. This still does not consider quality as test coverage could be 100% without the code working properly or efficiently.
5. **Code Quality:** Finally, there's code quality, this may be the most important and broad metric while being the hardest one to measure. This includes reliability, maintainability, readability, clear formatting and code repetition.

| Category | Technique |
|---|---|
| First Degree (direct involvement of software engineers) | Inquisitive techniques <ul style="list-style-type: none"> • Brainstorming and Focus Groups • Interviews • Questionnaires • Conceptual Modeling Observational techniques <ul style="list-style-type: none"> • Work Diaries • Think-aloud Protocols • Shadowing and Observation Synchronized Shadowing • Participant Observation (Joining the Team) |
| Second Degree (indirect involvement of software engineers) | <ul style="list-style-type: none"> • Instrumenting Systems • Fly on the Wall (Participants Taping Their Work) |
| Third Degree (study of work artifacts only) | <ul style="list-style-type: none"> • Analysis of Electronic Databases of Work Performed • Analysis of Tool Use Logs • Documentation Analysis • Static and Dynamic Analysis of a System |

(Lethbridge, Sim and Singer, 2005)

This taxonomy above shows examples of several these methods, it splits them into three categories First Degree which requires direct involvement of software engineers, Second Degree which requires indirect involvement of software engineers and Third Degree which requires the study of work artefacts only. There are trade-offs for each category, an ideal method would be easy to implement and adapt while giving us clear, accurate data that can be automatically measured without any manual work.

Computational Platforms

Once we decide what data to collect, the next task is to determine where and how to compute it. There are many software packages and companies that offer ways to collect, store and analyse data.

Personal Software Process (PSP) is a method of measuring the Software Engineering process and it was reviewed by Philip M. Johnson in Searching under the Streetlight for Useful Software Analytics (Johnson, 2013).

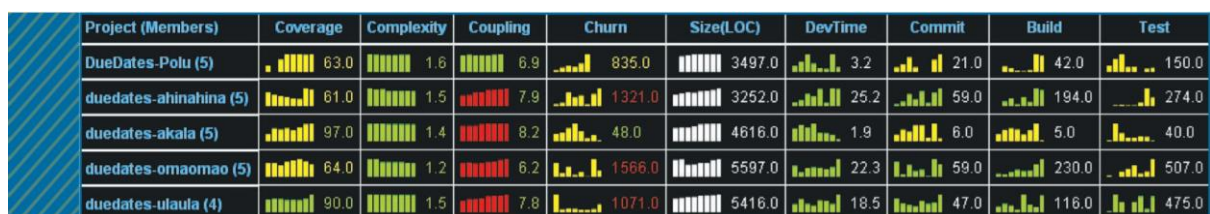
He reviews a version of the PSP which was created by Watts Humphrey in his book A Discipline for Software Engineering (Humphrey, 1995). In this method developers must fill out 12 forms, including a project plan summary, a time-recording log, a defect-recording log, a process improvement proposal, a size estimation template, a time estimation template, a design checklist, and a code checklist.

This is a mostly manual process, but since it's done in spreadsheets it is flexible and can be tailored to each specific use case. It is found that because of the manual nature of PSP there is a data quality problem due to misentry or subjectiveness.

In an attempt improve PSP Johnson investigated The Leap toolkit. Leap makes PSP more automated and normalises the data analysis. Leap is lightweight and portable as it creates repositories of data that developers can bring with them from project to project.

Unfortunately, the automation of PSP also comes with some costs, it has made certain elements of PSP harder to analyse and has also made PSP less flexible e.g. when creating a new analysis instead of just needed a new spreadsheet now you need to create a whole new Leap toolkit.

Next Johnson investigated Hackystat. Hackystat has client and server-side data collection, unobtrusive data collection, fine-grained data collection and personal and group-based development. It collects information automatically. It analyses much of the same information as PSP e.g. time spent on the project, size of the project, commits, but it also analyses some more advanced data, such as code coverage and complexity.

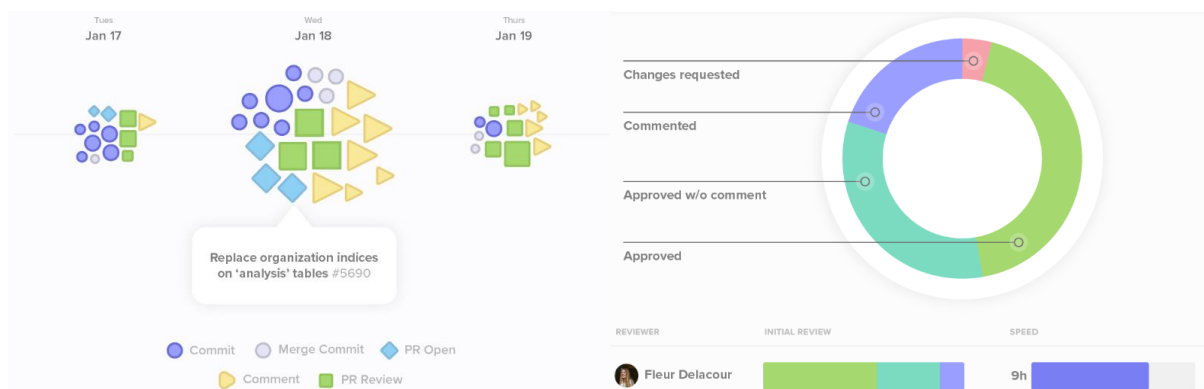


| Project (Members) | Coverage | Complexity | Coupling | Churn | Size(LOC) | DevTime | Commit | Build | Test |
|------------------------|----------|------------|----------|--------|-----------|---------|--------|-------|-------|
| DueDates-Polu (5) | 63.0 | 1.6 | 6.9 | 835.0 | 3497.0 | 3.2 | 21.0 | 42.0 | 150.0 |
| duedates-ahinahina (5) | 61.0 | 1.5 | 7.9 | 1321.0 | 3252.0 | 25.2 | 59.0 | 194.0 | 274.0 |
| duedates-akala (5) | 97.0 | 1.4 | 8.2 | 48.0 | 4616.0 | 1.9 | 6.0 | 5.0 | 40.0 |
| duedates-omaomao (5) | 64.0 | 1.2 | 6.2 | 1566.0 | 5597.0 | 22.3 | 59.0 | 230.0 | 507.0 |
| duedates-ulaula (4) | 90.0 | 1.5 | 7.8 | 1071.0 | 5416.0 | 18.5 | 47.0 | 116.0 | 475.0 |

The picture above is an example of a Hackystat output, it is easy to read so that managers/developers can get the information needed without having to know how the

program works or without having to spend unnecessary time searching through spreadsheets etc.

GitHub is an example of an open source platform that offers analysis and measurement for free. GitHub allows software engineers to host over 100 million repositories of specifications, design mechanisms, code etc. It is a web-based repository hosting service offering distributed version control and source code management. The platform provides the ability to measure many software engineering e.g. number of code commits/churns, number of contributors and frequency of contributor input. (GitHub, 2019)



Above is an example of Code Climate output (Codeclimate.com)

Another example of a company that offers data analytics for software engineering is Code Climate. It works in conjunction with Github to process mass amounts of data and provide the user with a clean interface where the user can clearly access code coverage, technical debt, progress reports etc. It supports many different languages, technologies and frameworks, as well as being known for its reliability, stability and test coverage. It also allows developers to visualise long term trends in the code base and compare overall quality of the project to others (Codeclimate.com).

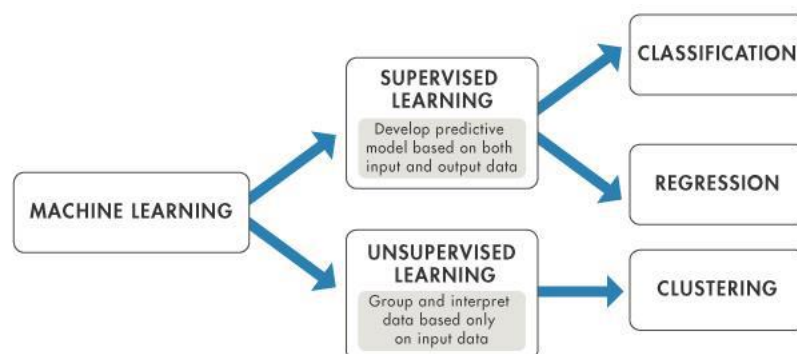
Algorithmic Approaches

When looking at algorithmic approaches to measure and assess software engineering processes we can split these approaches into two main groups; supervised and unsupervised machine learning. (Brownlee, 2016)

Supervised machine learning is when the algorithm used has an input and an output the machine learning means that the more inputs the algorithm has the better it can predict the output.

Unsupervised learning differs as it only takes input values and there is no output value. Instead of outputting a predicted value unsupervised learning uses it to create models to discover some fundamental structure.

Below is a simple diagram showing supervised and unsupervised learning and their subcategories.



Supervised learning has two main subcategories Classification and Regression.

Classification consists of outputs that are categorical variables such as language type whereas Regression consists of output that are numerical variables such as test coverage.

Unsupervised learning also has two main subcategories clustering and association. A clustering problem is where you want to discover the inherent groupings in the data, such as grouping engineers by coding languages. An association rule learning problem is where you want to discover rules that describe large portions of your data, such as people that code X language efficiently also tend to code Y language efficiently (Brownlee, 2016).

The main example of supervised learning is linear regression. It's made of a linear equation i.e. there's a coefficient for each input value as a scale factor, and another coefficient as the y-axis intercept. There are several different techniques used to estimate these coefficients and prepare models. I will talk about two.

Simple Linear Regression is the first method, which only uses a single input. Statistical properties from the data e.g. mean, standard deviation, covariance and correlation can be

used to make a prediction of the value of the coefficients. This method is not widely used due to its simplistic nature.

When there is more than one input value, Ordinary Least Squares can be used. This method calculates the distance from each data point to the regression line or line of best fit, squares all these distances, and then sums them together. This sum is the sum of squared errors, and Ordinary Least Squares aims to minimise this. The coefficients which have minimal value i.e. closest to the line should be used. Other methods of linear regression include Gradient Descent and Regularisation. (Brownlee, 2016)

Now I will move to an example of unsupervised learning, which is K-means clustering. The aim of cluster analysis is to establish if there is a group structure in a dataset, if so, how many exist and what are their individual structures. The first step in K means is to specify the number of clusters K , and randomly assign each data point to a cluster. Once the clusters are created, the cluster centroid must be computed. Then, re-assign each data point into a cluster based on which cluster centroid it is closest to. Recalculate the cluster centroids for the new clusters and repeat until no improvements are possible.

Principal Components Analysis (PCA) is another good example of an unsupervised learning technique. It becomes useful when we want to compare large sets of data to find sets of correlated variables using linear combinations.

PCA's goal is that most of variability will be explained by the first few principal components, allowing us to figure out which data measured from an engineer's performance has the most effect on their productivity. Lower principal components provide less explanation and can be considered inherent noise. If we assume all data we collect has a strong impact on performance and use all of it as input into machine learning algorithms, we could be modelling with inherent noise.

PCA is a measure of how each variable is associated with one another (Covariance matrix), the directions in which our data are dispersed (Eigenvectors) and the relative importance of these different directions (Eigenvalues). PCA does however make independent variables less interpretable (Brems, 2017).

Ethics

The EU General Data Protection Regulation (GDPR) is the most important change in data privacy regulation in 20 years, its aim is to protect all EU citizens from privacy and data breaches (Eugdpr.org, 2018).

GDPR applies to any company processing personal data of EU citizens. Companies in breach of GDPR can be fined up to 4% of annual global turnover or €20 Million (whichever is greater). Companies that are aware of a breach must notify their customers and the controllers within 72 hours of finding said breach.

Companies are no longer able to use long illegible terms and conditions. The request for consent must be given in an intelligible and easily accessible form, with the purpose for data processing attached. It must be easy to withdraw consent. Companies must also state where and for what purpose are they requesting the specific data. This allows for more transparency for data subjects. The data subject also has the right to have their data erase and data that is no longer relevant to its originally stated purpose must be erased. Finally, GDPR states that companies must include data protection from the onset i.e. companies may not collect data until they have the infrastructure to deal with and protect it (Eugdpr.org, 2018).

An example bad ethics in data collection and analytics is Cambridge Analytica who were hired by Trump's 2016 election campaign for research and advertisement. They gained access to private information on more than 50 million Facebook users and used this data to identify the personalities of American voters and influence their behaviour. The data included details on users' identities, friend networks, likes and even locations. Facebook prohibits this kind of data to be sold or transferred "to any ad network, data broker or other advertising or monetization-related service." But at the time insisted that it was not a data breach since Cambridge initially had access to the data for academic purposes. The data, or at least copies, may still exist (Granville, 2018).

The ability to collect and analyse data is moving much faster than any legal and ethical guidelines can manage, this is where the general ethics of a company comes into play. Companies may take the approach that while a software engineer is at work, working on their hardware that they should have access to all data that may come from this. I believe this is a very narrow view and does not take into consideration the wants and needs of the software engineer and can therefore also negatively affect the company. Being under constant measurement and assessment can be extremely stressful for an employee and this alone would hinder job performance and may also contribute to mental health problems.

When measured incorrectly this may not only skew the perceived employee performance but may also discourage creativity as the employee will be more focused on reaching certain goals rather than completing their assigned task to the best of their ability. I believe it is also a naive and simplistic view to consider not availing of the ability to accurately measure and assess employees so it should be then modelled with consideration of the employee, taking into account how it may also benefit the employee for example a software engineer could be told in what area's their coding skills are lacking and maybe how they could improve them or refer them to a software engineer in the company who has strengths in these areas. Finally, transparency should be of the utmost importance when it comes to considering the employee as this allows them to understand how and why they are being measured which will alleviate some scepticism and wariness.

Conclusion

The ability to measure and assess software engineering processes is still a relatively new and ever-expanding concept. There are many facets of software engineering that need to be considered when attempting to measure it. It is not as simple as measuring for example a salesperson's sales numbers or an athlete's endurance. This creates a very difficult task to accurately assess while taking all areas into account.

Once you have figured out what areas to access it is also very difficult to figure out how to access them. Ideally, these processes would be measured in the background whilst not disturbing the software engineer or inhibiting their workflow. There are packages available for this but not all may be as accurate as you need, and some may be seen as unethical.

Ethics may be one of the most overlooked and undervalued section of data collection and assessment. The main reason why a company might want to measure software engineering processes is to improve their output both in terms of efficiency and quality, but when ethics is not considered this can greatly affect an employee's performance, in my opinion the process of measuring and assessing a software engineer must be a transparent one that the employee agrees with and can benefit from.

Bibliography

1. Lethbridge, T., Sim, S. and Singer, J. (2005). Studying Software Engineers: Data Collection Techniques for Software Field Studies. pp.311-341. Available at: <https://link.springer.com/content/pdf/10.1007%2Fs10664-005-1290-x.pdf>.
2. Johnson, P. (2013). Searching under the Streetlight for Useful Software Analytics. pp.57-63. Available at: <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6509376>.
3. Humphrey, W. (1995). A Discipline for Software Engineering.
4. York, B. (2015). The Best Developer Performance Metrics —. [online] Medium. Available at: <https://medium.com/@yupyork/the-best-developer-performance-metrics-6295ea8d87c0>.
5. GitHub. [online] Available at: <https://github.com/about>.
6. Codeclimate.com. Understand Each Step of Software Development Life Cycle | Velocity. [online] Available at: <https://codeclimate.com/velocity/understand-diagnose/>.
7. Brownlee, J. (2016). Master Machine Learning Algorithms. 1st ed. Available at: <http://index-of.es/Varios-2/Master%20Machine%20Learning%20Algorithms.pdf>.
8. Brems, M. (2017). A One-Stop Shop for Principal Component Analysis. [online] Medium. Available at: <https://towardsdatascience.com/a-one-stop-shop-for-principal-component-analysis-5582fb7e0a9c>.
9. Eugdpr.org. (2018). [online] Available at: <https://eugdpr.org/>.
10. Granville, K. (2018). Facebook and Cambridge Analytica: What You Need to Know as Fallout Widens. [online] Nytimes.com. Available at: <https://www.nytimes.com/2018/03/19/technology/facebook-cambridge-analytica-explained.html>.