Measuring the Software Engineering Process

Name: **Rafael Mendes**

Student No. : **17330951**

Introduction

As more and more data is available to us and as methods to gather, manipulate and interpret this data also improve, it is important to think about how useful these large sets of data are and what we can achieve with them. Data gathered during the Software Engineering Process is no exception and in fact, it is not new to the world of data driven improvements. From quantity of code delivered to quality ratios, management in software companies have been using these metrics to increase productivity and cut on any waste of resources for many decades now. In this report I will outline the different metrics that are commonly used today, that platforms which gather these metrics, algorithmic approaches for gathering these metrics, and the ethical concerns associated with these large data sets.

Measuring and Assessing the Process

In my opinion the Software Engineering Process can be divided into two main parts, programming and coding a feature and then delivering said feature. I believe this distinction needs to be made as the data which can be gathered during both phases is slightly different. When we talk about the first phase we're mainly interest in productivity i.e. the volume code an individual can write and how long it takes for them to do so. However once we get to the code delivery phase we are more interested in code quality, how long it takes for bugs to be fixed or code to be refactored, test coverage, how successful the feature's adoption is, etc. As such this section of the report will be divided into two parts, measuring and assessing code quantity and measuring and assessing the code quality.

Quantity

The most basic unit used in the industry to measure volume of code produced by an individual is the “LOC” or Lines of Code. This tried and tested method has been around since the late 1960’s and is widely used as a basic unit when calculating more complex metrics for example number of faults in individual modules in a codebase. The reason for its success and longevity as a metric can be tied down to two things, how easily it is to compute and how easy it is to visualise. Furthermore it’s easy for anyone to understand and an exact as well as objective measure of quantity of code written. However with the advent of the Agile methodology adopted my most software engineering teams nowadays, LOC begins to be left behind as it is not as valuable as other metrics when applied to a whole team rather than just an individual.

When talking about team-wide metrics especially for teams which operate in an Agile Scrum environment, the most widely used and popular metric is Velocity. Velocity is defined as the number of stories completed multiplied by the story points of each story completed. If done correctly this is a very good metric has it is able to combine both quantity of work done with the complexity of this work something the LOC measurement fails at. However this metric is very much dependent on the team and mangers as they are the ones who define the stories and story points, a team who does not have a good grasp of Agile methodology or one who is unwilling to engage in it will find this metric near useless. Furthermore to outside members of the team, these metrics are unclear and perhaps difficult to explain, however as long as the team managers understand them correctly they can be used and interpreted to cause big improvements and necessary changes.

In conclusion when discussing quantity measurement there are two key metrics available, Lines of Code written by an individual and the Velocity of a team during an Agile Sprint. Those metrics when used with time elapsed (e.g. weeks, months, quarters, etc.) can paint a very clear picture of an individual’s and a team’s productivity.

Quality

Once the code has been delivered and shipped to a production environment there are two key metrics teams can use to measure its quality. The first is the “Mean Time Between Failures”, this is a very important metric and something teams should be keep a very close eye on. A team tracking this metric and communicating it to its members will end up more motivated to deliver quality code as they will want to keep this statistic as low as possible. Any team who chooses to ignore this is choosing blissful ignorance and will find it much harder to quantify the quality of their product and as a result will also have difficulties delivering impactful improvements. This metric is also pretty simple to calculate and easy to visualise and has direct correlation to quality of code making it a very strong candidate to be taken into account.

The second metric “Mean Time to Recover/Repair” is quite similar to the first one, however in my opinion it provides even more information. When this statistic is low it can imply two things, the first is that the code is of such high quality it rarely breaks and as such it is rarely in need of repair. The other equally interesting implication is that if your team is fast at troubleshooting, this time will be equally low also revealing the productivity of the team. The “two in one” value gained from this metric is very powerful and again when shared with a team of developers will motivate them to work harder to keep this metric at a low amount.

In conclusion using both of these easy to measure metrics, delivering quality software can be somewhat gamified, as the team strives to get better and better scores for both of these. Not only that but they also paint a very clear picture of the quality of code produced allowing managerial staff to make better decisions when faced with customer complaints.

Conclusion

The metrics mentioned above are all popular and widely used to measure productivity of both teams and individuals and quality of code. They are largely objective and relatively easy to calculate, manipulate and visualise making them very versatile. However especially when it comes to productivity I think some improvements can be made. In my opinion the

metrics outlined above are not good enough to identify key developers (i.e. the 10xers), they simply calculate volume rather than actual impact. Personally what I would be interested in measuring, is how much developers engage with other developers in a software engineering company. In particular, it is very possible to interrogate the GitHub API to find out which developer is engaging the most with their team's Pull Requests. To me the MVP of a developer team is not only the one who is most productive and can push out the most features, it is the developer which can do that and also elevate the overall team's level through constructive criticism, by being helpful and sharing their knowledge.

Computational Platforms

As more and more software companies look to data and metrics to bring about change and improvement in their teams, the number of platforms that provide this service has increased accordingly. One such example is GitPrime, a service which "aggregates historical git data into easy to understand insights and reports". Using this service, teams have access to a large quantity of objective data which would allow them to identify bottlenecks, worrying trends and overall keep track of the health of their teams. More specifically the product can be partitioned into three key areas each with their specific services.

1. **Measuring and comparing Agile Sprints**

- Work Log - Records team member's activity during a sprint
- Project timeline - Allows users to see what effects key events (e.g. addition of new testing tools) had on the sprint's velocity
- Retrospective – Used to evaluate the success of releases and compare different sprints (especially good service if a team holds any kind of retrospective meeting at the end of each sprint)

2. **Managing Pull Requests**

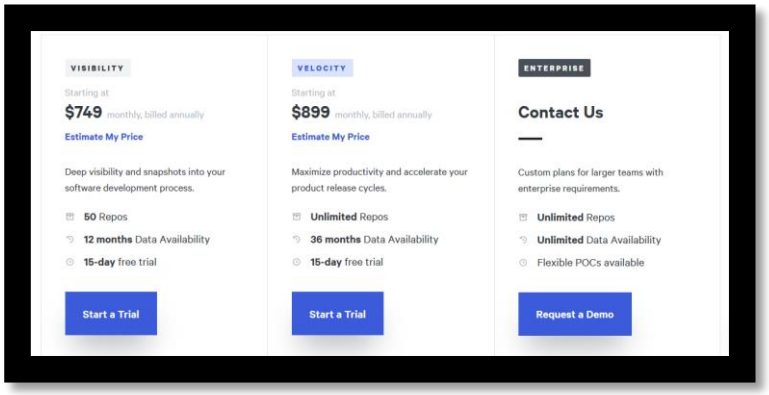
- Workflow – Great visualisation of data pertaining to Pull Requests such as unreviewed and old PRs as well as PRs that are causing a lot of discussion in the form of comments

- Resolution – Measures metrics such as Time to Resolution and Number of Reviewers for Pull Requests and keeps these so that they may be used for comparison purposes over a period of time

3. Teamwork Dynamics and Culture

- Collaboration – Provides multiple metrics such as percentage of Pull Requests or commits that get zero engagement, percentage of team involved in feedback and number senior engineers providing feedback or mentorship
- Knowledge Sharing – Provides data on knowledge silos within a team, i.e. which individual has the most amount of knowledge and should be sharing it

GitPrime proves that there's potential for valuable information gathering from something as simple as the GitHub API. Not only that but they have big clients which see the massive value in having easy access and good visualisations of this information. Take for example a testimonial from a Tesla Engineering Team Lead who says "GitPrime metrics have completely transformed the way we think about productivity". Furthermore we can infer the value of the product based on their prices, since the cheapest plan is priced at $749 per month, it becomes very clear how valuable companies think this product is.



If a team cannot afford the monthly prices of GitPrime then there are a free alternatives such as Hackystat. Hackystat is an open source framework which allows for collection, analysis, visualisation, interpretation, annotation and dissemination of software development process and product data. The way this framework is able to collect all this data

is by attaching itself to development tools such as IDEs and Text Editors and collecting “raw” data which is sent to the Hackystat SensorBase for storage. This SensorBase can be queried by other web services to acquire higher level abstractions of the original “raw” data. As a result this framework is quite powerful and can be easily integrated with other web applications and services. However it is no longer in active development, even though it is still usable.

The final alternative is for the individual to query the GitHub REST API themselves to gather the information they are looking for. This is quite simple and achievable in a large number of programming languages using a myriad of libraries, however the difficulty comes in manipulating the large amount of data returned in order to get some useful metrics. Another drawback is that there is a rate limit (5000 for authenticated users) of requests which limits what you can do with this simple method. However it is also free and available for everyone to use and it can also serve as a good introduction for software process data gathering and manipulation.

In conclusion, it is clear that there are many platforms which allows us to get valuable information about the software process. The nature of each product is also varied ranging from paid to premium and from appropriate for start-ups to enterprise level. As a result it is more than likely that you will always find a platform to meet your needs and/or budget.

Algorithmic Approaches

Large data sets are not difficult to gather, the challenge comes in creating meaningful metrics from them. This is where certain algorithms and methods come into play. One such set of rules is Halstead’s Metrics, a series of rather simple equations which have been used by researchers “to evaluate student programs and query languages, to measure software written for a real-time switching system, to measure functional programs, to incorporate software measurements into a compiler and to measure open source software”. These equations range from calculating the length of a program all the way to calculating the required programming time. This set of equations can all be derived from four parameters which are:

- n_1 – Number of distinct operators
- n_2 – Number of distinct operands
- N_1 – Total number of occurrences of operators
- N_2 – Total number of occurrences of operands

Although these equations can be used to calculate a varied set of valuable metrics, they are not widely used and are by no means industry standard. This is largely due to the fact that there is no general and specific rule for identifying operators and operands in any programming language. Instead someone who wants to avail of these metrics will have to use examples and intuition which are simply not enough to get accurate and worthwhile results. Furthermore there is large possibility that two people working with theses metrics come up with different definitions for operands and operators, further showing how inconsistent this algorithmic method can be. In conclusion since Halstead himself has not "provided a clear and complete counting strategy to distinguish between the operators and the operands", this method remains largely unused by companies but still a great topic for research as it has the potential to be truly powerful and ground-breaking.

Another algorithmic approach involves the use of Bayesian networks and activity-based quality models, to assess and predict software quality. Bayesian networks are a kind of probability based graph that uses Bayesian inference for probability calculations. The edges in this directed graph represent cause-effect relationships while the nodes represent unique variables each with their own node-probability table (NPT) which define relationships and the uncertainty of each corresponding variable. This a great model for quality prediction and assertion, as it allows for corrections to be made over time when more information becomes available (i.e. when there are less unknowns). When these graphs are combined with Activity-Based Quality Models which are used to create guidelines and quality targets, their quality of prediction and assessment is enhanced even more. Although not extremely popular in industry, these methods have been showed to work even when dealing with very large amounts of data for example those provided by a real NASA Tomcat servlet container.

Finally, once can always extract many kinds of data from the GitHub API programmatically and then manipulate to for example calculate an average over a period of time. This can be

achieved in many different ways with many different programming languages, especially given the amount of libraries that exist for making requests to the GitHub API. However, since you are limited by the rate limit, this method may be slow at times and it also means that every request counts so requests wasted on useless information are very expensive.

In conclusion, given the varied algorithms and methods to create metrics for the software engineering process, it is hard to argue that this process cannot be measured.

Ethical Concerns

There are many ethical concerns which arise when one considers the large amounts of potentially personal data and the possibly intrusive methods used to gather it as part of creating metrics for measuring the software engineering process. The first concern one should have if they have signed off on their company gathering data about them, is knowing where this data is stored and more importantly knowing how safe it is. When your data is being constantly gathered it is important to know where exactly it is being held and whether there has been any breaches of it. Furthermore when leaving the company you must be able to have that data deleted, otherwise those who are holding your data are in breach of EU GDPR Law. As shown by multiple successful data attacks carried out against huge companies like Google and Facebook, your data is not guaranteed to be safe and this concern should always be present in your mind. Moreover if your employer insists in gathering your data but does not share where this data is stored, what it is used for and whether it is safe or not then there should be a large concern as those questions should have clear answers.

Another ethical concern one should have, is whether this kind of constant monitoring causes stress and mental health problems in the people being monitored. It is highly possible that when this data is gathered and presented to an individual, it can be used in negative ways in order to get more “value” (for example work hours) from them. If a company is successful in maliciously using this data to increase their workers’ productivity past a healthy amount, it will undoubtedly cause physical and psychological harm on their employees. This a major concern however it is offset my worker’s rights which in theory should protect employees from being overworked by their employers.

A further concern would be the gathering of data outside of work hours i.e. during coffee/lunch breaks. Although some may argue that data of this nature can be extremely useful when creating metrics for company/team wide dynamic and culture it's still very personal data that few would be comfortable giving. Furthermore as mentioned previously this data would be gathered technically outside of company time where what the employee does should have little to no impact on his employer's perception of him. Monitoring employees in this way outside of company hours and outside of company buildings should not be allowed unless the employee themselves specifically condones this action. Moreover there is a possibility that if the employer does it get access to this data, it is misused or used maliciously. With the kind of sensitive and personal information an employer would have access to about their employees, the potential for psychological manipulation and bullying would surely increase.

A final ethical concern that I would like to bring up is how the employer could be able to leverage their power to force their employees into signing over their data. It would be completely immoral and unethical if once an individual was offered a job at a software engineering company, they were asked to leave unless they allowed their employers to monitor them in a myriad of ways. Another variation of this would be if promotions or key positions in a company were unreachable unless the employee again allowed the company to monitor or if they are already being monitored to allow the company to increase monitoring efforts. An employer leveraging a job or promotion to gain something from the employee/potential hire should be completely illegal.

In conclusion although there are many ethical concerns associated with gathering and monitoring employees to further understand the software engineering process, that are most definitely an adequate number of laws in place which would prevent the employee from being mistreated. However these laws will not protect the individual if any kind of legal agreements are signed so it is important that an employee reads and understands these clearly before joining any company. Moreover it is also very important for companies to keep these documents clear and to not purposely or accidentally obfuscate any sort of important detail or clause.

Conclusion

The importance of being able to measure and understand the software engineering process has been made very clear in recent years. However methods to do so continue to be somewhat basic and focus solely on work produced through code. More and more metrics are being continuously added and investigated but many are still in their infancy and require many more years of research before they become relevant. Yet as this exciting research and improvement continues we must not lose sight of the ethics which surround it, so those must be discussed alongside it, with laws being readily made in preparation for the increase interest in this data. Arguing that the software engineering process cannot be measurable is simply impossible nowadays as there are many examples of successful platforms that offer this service. However I would make the argument that these metrics have not reached their full potential yet and that they only reveal and shine a light on a smaller part of a much bigger picture.

References

Fenton, N. E., and Martin, N. (1999) "Software metrics: successes, failures and new directions." Journal of Systems and Software 47.2 pp. 149-157.

https://techbeacon.com/9-metrics-can-make-difference-todays-software-development-teams

https://klevas.mif.vu.lt/%7Esigitas/Kokybe/Straipsniai/1-s2.0-S0950584910001175-main.pdf

http://profs.etsmtl.ca/aabran/Accueil/AlQutaish-Abran%20IWSM2005.pdf

https://www.gitprime.com/

https://www.businessinsider.com/tracking-employees-with-productivity-sensors-2013-3?IR=T

https://hackystat.github.io/

https://developer.github.com/v3/

https://towardsdatascience.com/introduction-to-bayesian-networks-81031eeed94e