The Measurement of the Software Engineering Process

Table of Contents

Introduction

The software industry is not only among the newest of recent industries to surface in the modern era, it is also one of the fastest growing sectors in our current disruptive age of computer based technology. Worldwide spending on information technology is estimated to reach near $4 trillion in 2019 according to Gartner[1], a 3.2 percent increase from 2018. Companies in every single industry in the world are being flipped upside down by the radical changes and innovations technology has been bringing to the table. IT has travelled past the point of being a "useful platform that enables organisations to run their business on", and it is arriving at a reality of being "the engine that moves the business".

With these profound changes to the way businesses are run, the underpinning technological practices that they rely on have come under the microscope. A Stripe study has revealed that access to developer talent has become more important to companies than access to capital. With developer salaries as high as they are, and entire businesses at risk of going bankrupt should things go wrong, there needs to be some concrete way to measure all aspects of the software development process. Talent recruitment with a thorough interview process is not enough on its own anymore, there needs to be ways to evaluate and moderate a developer team on the efficiency and value of their work.

There has resultantly been a myriad of metrics and trends being tracked and used to make managerial decisions with relation to coding teams and their projects, some of these have been beneficial, others detrimental to either the projects or the developers themselves. Some ethical issues becomes apparent when deciding what metrics to use, *and in what way to use them.*
In this report I will explore in detail the methods of software development evaluation, the difficulties involved within them and the ethical issues that have been raised as a result.

Industry's Shift in Focus to Software Efficiency

Despite software development being the target of massive worldwide spending, and despite its profoundly groundbreaking changes to countless aspects of human life, it is still very much an industry in its infancy. With the first official software company entitled *'Computer Usage Company'* being founded only 64 years ago in 1955[2], the software development process has since burgeoned into an entity of unprecedented scale. Software has become the focus of companies of every kind, not just companies where software is the primary intended product. Every company has had to make the transition to online services and in doing so have started to face a new range of issues, from security vulnerabilities to complex governmental regulations,which has created a resulting large demand for programmers. However the amount of capable developers is scarce. The *US Bureau of Labour* estimates that there will be 1.4 million developer job openings in the United States, but there will be only 400'000 computer science graduates to fill them[3]. With developers in such high demand, highly competent and skilled developers have become an even more valuable commodity.

This strong demand for developers, as well as the explosion of importance of software for companies to compete and thrive, has led to software development and maintenance being a very costly endeavour. With so much money paid to developers each year and so much weight behind the impact of their work, there has consequently been a necessity for companies to *shift focus* to the pursuit of *efficiency* from their developers. According to a survey conducted by 'Stripe', a whopping 42.2 percent of time is spent inefficiently by the average developer.[4] 33 percent of total time is wasted on technical debt (this is essentially the maintenance of outdated legacy systems). The remaining 9.2 percent of inefficiency can be chalked down to bad code, which can be a result of a whole host of factors, including but not limited to management's poor prioritisation of tasks, a necessity to build custom technology when pre-existing purchaseable

technology for the same function could be used, or simply due to an incompetent programmer.
The road to high efficiency is by no means an easy one, software development is a relatively new practice, it is highly complex and often vagarious, it is hard to know where to even start when trying to improve efficiency of the process, which neatly brings us to the topic of this report: what metrics are there to assess the software development process both on a collective and individual level, of these metrics, which are the best at indicating code efficiency or lack thereof. Most importantly, in what way should companies use the results of these metrics in order to improve the quality and productivity of the software development process, and what ethical implications does the use of these metrics entail.

Difficulties in the Evaluation of the Software Engineering Process

---

How do we measure code? As I have already mentioned, it is far from a trivial matter. Do we count up the lines of code to ascertain their worth? This is certainly one method, but we find that functionality that is achieved in 1000 lines of code by one programmer can often be achieved in less than 100 lines by a more experienced programmer, or even the same programmer using a more elegant approach. The same can be said about the number of commits in the context of a code repository , while committing frequently is good practice, it doesn't indicate anything to do with the quality of code.

In general, measuring anything to do with the coding process is incredibly difficult. Software by nature is unlike any other product. For example when building a bridge you can say with certainty how much of the bridge is completed and how much of it still has to be constructed, software on the other hand is much to the contrary, for the vast majority of development it is really hard to know how much work there is left to be done and how long it is going to take. Things can and do happen during the development process that require unforeseen extra work or changes to pre-existing code, for example, subsystem incompatibilities, bugs, hardware

insufficiencies, or in many cases just not realising that a task was going to take way longer than it was initially projected to. As the famous quote said by Tom Cargill at *Bell Labs* goes,
-"*The first 90 percent of the code accounts for the first 90 percent of the development time. The remaining 10 percent of the code accounts for the other 90 percent of the development time.*"

No matter what you do these problems can arise, however, in the all too common case that a project is insufficiently planned and programmers are made to start coding before having a concrete blueprint of how the finished product is meant to look, development time and the likelihood to encounter problems further down the line is increased substantially. Sadly this is a very common scenario in the industry. For most software projects the ideal first step is a process involving just a few software architects, they should take the necessary time to make a detailed design of the system or program. During this process the large body of developers employed by the company might have to sit idly by until the planning is complete, which in turn infuriates stakeholders who then demand the coding begin right away. A lack of understanding and respect for the coding process will cause problems to arise more frequently, and the problems will in turn require more time to fix.

While it is true that there can be many difficulties and uncertainties when it comes to analysis of software production, that does make it any less important. With so much capital and so many jobs at stake in the current industry climate it is *crucial* that we clearly define metrics of analysis, develop them to a standard of accuracy, and rank them in order of their importance to the projects they are applied to. Care must be taken to ensure these metrics are used ethically so as not to encourage programmers to attempt to cheat the metrics, and so as not to create a toxic engineering culture where people are rewarded for the wrong things.

Metrics of Analysis

We can generally split most software development metrics into two categories:

1. Engineering Process Metrics

or

2. Software Performance Metrics

Engineering process metrics measure how the product is being built and the team's productivity. Software performance metrics measure the delivered product, this might be in terms of application response time or a measure of customer satisfaction. I will outline some of metrics from each of these categories and discuss where they might be of use.

Engineering Process Metrics

Ticket close rates - the rate of the amount of tickets completed by the team or a contributor, tickets can be the completion of a subtask or the fixing of a bug. Using this metric comes with the assumption that you have split up the development process into tasks of relatively equal size, which is generally not the case as unforeseen issues and delays can arise. This metric can be used to identify if a certain developer is stuck on a particular task and can allow management to delegate another team member to the task if needed. It should not be used to evaluate individual performance, but can be used to roughly gauge the speed of the team over time.

Code Churn - a measure of the extent to which code in a project is evolving, it is generally measured by taking the number of lines modified, deleted and added in a certain time period, and dividing it by the total lines added. It gives a picture of how much time is being spent altering existing code as opposed to writing new code. Code churn varies a lot from project to project but also changes throughout the course of the project. There are

many causes of code churn, when bugs need to be ironed out or when sections of code need to be overhauled or rewritten, code churn will be high. High code churn can also indicate if certain tasks are proving difficult to developers. When prototyping or polishing for a release code churn will tend to be high, however if code churn is still increasing while a deployment deadline is approaching it is not a good sign, you want the opposite, you want to stabilise more and more code as you approach a deadline to minimise post release defects. Code churn trends over long periods can be useful for this reason, you might want to postpone a deployment deadline if code churn is still very high as it indicates volatility in the codebase.[5]

Refactoring Rate and New Work Rate - In the same vein as code churn are these two metrics. Refactoring rate is a measure of how much old code, that hasn't been altered in a while, is being altered, such code is often called legacy code and inevitably needs to be changed to be compatible with newer additions to the system or changes in hardware. New work rate is the measure of all the new code being added. It is useful to measure these as ongoing trends to ensure there is a reasonable balance between the amount of new features being added and old features kept up to date. It's bad to have technical debt but it's worse to have a stagnant product.[6]

Cyclomatic Complexity - This is the measure of code complexity, it is a quantitative measure of the number of linearly independent paths through a programs source code. In order to calculate code complexity, we can disassemble code into its assembly language equivalent and represent each instruction as a node on a graph, connect an edge between a node and all it's possible branch instructions. We then simply count up the nodes and edges and plug them into the formula M = E - N + 2 which gives us it's complexity value.[7] This metric can be used in software testing to determine the amount of test cases needed to fully test a program. On the other hand, cyclomatic complexity testing generally provides no benefit from a managerial standpoint.

Lead Time and Ticket Response Time - Lead time is the time period between beginning a projects development and it's delivery to a customer. It can be used in gauging a team's speed and providing an estimate to the customer of how long development will take, accurate estimates are made by taking into account the nature of the project and the team's performance on similar projects in the past. Ticket response time is the time between the creation of a ticket and the beginning of work on the ticket. Having a low ticket response time is important if you mean to implement continuous delivery of a product and will impact customer satisfaction significantly. In some cases ticket response time is quite crucial to a service, for example a server engineer might receive a ticket about the failure of a server due to an increase in traffic, the time that ticket sits idle without reaction is time that a whole service is totally out of action, quick response in these types of situations is needed, and ticket response time is a good metric to try and improve on.

Software Performance Metrics

Customer satisfaction score - It is very valuable to collect data from either your client or your customers based on their interactions with the development team or just their use of the software itself. In the former case, seek to collect a score of customer satisfaction with certain aspects such as the teams clear and open engagement with the needs of the client, the frequency of prototypes to allow the clients a more ongoing say in the production. In the latter case of third party software testing, you can ask testers to score software based on usability, intuitiveness, user interface, response time, and quite a few others. Such data can be crucial in evaluating where the software excels and where it might fall short.

Test Coverage Ratio - This is a metric indicating how many of the total lines of code in a project are being tested by unit tests. The generally accepted "sufficient" coverage ratio hovers around 80% but for crucial systems it higher. Keeping a track of this metric over time can help ensure quality isn't being traded off for velocity.

Meantime Between Failures and Meantime to Recover - This metric is a test on how the software performs in a production environment, software failures are unavoidable so it's good to keep track of how often they occur and how well the software recovers and preserves data. If the meantime to recover grows over time it is an indicator that developers are becoming more effective in understanding issues and how to fix them.

Ethical Implications in the Tech Industry

Software engineering has become one of the crucial pillars for organisations to perform well in their industries, as a result of this, software engineering efficiency has become the biggest challenge to companies, even above access to capital.[8] This importance being placed on software development and the need for metrics can lead to some very unethical practices in the industry, almost always from a lack of understanding of the nature of the process. While performance metrics can be very helpful in the maintenance and motivation of a team, when used incorrectly, they can be massively detrimental to the team's efficiency, attitude and mental health.

Teams that feel like they are being unfairly tested on metrics, of which the nature is unknown to them, will grow to become understandably disgruntled, it is important for a level of transparency to be used by management with relation to the metrics they use, so as to create an element of trust with developers. If a metric is such that a developer can 'cheat the system' by paying special attention to bolster that metric, then it is most probably not a good metric.

It is doubly important to ensure that the metrics that are chosen are the right ones, with all these metrics available to us, we must decide on the ones that are most important for business goals. "The more metrics you track with the same level of importance, the less importance you give them".[9] Management should involve the software teams in deciding which

metrics have the most relevance for a certain task and should therefore be given the most importance.

Business success metrics should drive software improvements, not the other way around. Metrics should not be looked at as just an indicator of how well development is going, to get the full value out of these metrics you must use them to improve on shortcomings, set milestones and achieve production goals for a team in a healthy, and optimistic manner. A software development team's good mental health and a trust in their management's approach and understanding goes a long way. It improves productivity as well as their own optimism on the project, as opposed to them becoming indifferent to their project's outcome and quality.

It is important to firmly understand the limitations of these metrics. You shouldn't use them to make any serious judgements or decisions on an individual's performance in a team. Management is always difficult and contextual, without discussing an issue in detail with the team there is no way of truly understanding the nature of the said issue. Metrics should only be a tool of making inquiries to understand what really happened and by result, get a grasp of the intricacies of the team, the individual members, and the best practices for managing the team.

Conclusion and Summation

In summation, software development is an extremely complex and often ambiguous process. It requires management to take a hands on approach in gauging the efficiency of a team. Such an approach can be taken with the assistance of a handful of metrics which can be used to collect data for a variety of different purposes. This data should be used only as a conversation starter and never as a decision maker due to the nature and nuances of software engineering. When used properly, the clever use of these results can have a crucially beneficial impact on the productivity of developers, the synergies of a team, and the satisfaction of clients, customers and stakeholders. Care needs to be taken to develop these

measurement practices in an ethical and healthy way so as not to jeopardise the mental health of programmers and to avoid the creation of a totally toxic programmer culture.

References

---

https://www.gartner.com/en/newsroom/press-releases/2019-01-28-gartner-says-global-it-spending-to-reach--3-8-trillio

https://ieeexplore.ieee.org/document/279238

https://www.techrepublic.com/article/more-women-developers-hell-yes-says-holberton-school/

https://stripe.com/files/reports/the-developer-coefficient.pdf

https://anaxi.com/blog/2018/11/05/engineering-efficiency-is-now-the-biggest-challenge-to-companies/

https://codescene.io/docs/guides/technical/code-churn.html

https://anaxi.com/software-engineering-metrics-an-advanced-guide/

https://www.froglogic.com/blog/tip-of-the-week/what-is-cyclomatic-complexity/

https://hackernoon.com/how-to-use-and-not-abuse-software-engineering-metrics-3i11530tr