Vladimir Ustimenko
17306631

Measuring Engineering - Report

Introduction:

As software industry exploded in size since the first computers became commercially viable some 60 years ago, so did the need to quantitatively measure exactly what is being done, and how efficiently. What in the past could be achieved via empirical means has by now been superseded by more ‘scientific’, rigorous approach. It incorporates various quantitative metrics that evolve as programming paradigms and conjuncture . They vary from company to company but always have a profound impact on the company and the programmers alike and it is important to understand what these metrics are, before delving into the ways in which we can measure them, to ultimately examine the implications for individuals and employers from an ethical point of view.

Metrics:

The fundamental purpose of metrics is to measure productivity. In general terms productivity is defined as the amount of output per unit time. In most productivity metrics are a fairly simple concept to apply: surgeons’ sole job is to save lives, salespersons’ - to sell company’s produce. So the former’s main metric could perhaps be the number of lives saved and the latter’s – cash value of product sold. But even with these relatively obviously measured professions, various other metrics may be implemented. Which surgeon is better: a general surgeon who operates on 100 patients in a month and loses 1 or a brain surgeon who saved all 10 of theirs? The answer to this question requires careful ethical and moral considerations but it goes to show just how complex a topic metrics are.

Metrics Criteria:

1) **Consistency:**

A good metric must provide a consistent result for a prolonged period of time. One of the most useful applications of metrics is examining engineer’s historical performance for trends that provide the company with feedback regarding which of their approaches work and which do not. Comparing multiple engineer’s past results could inform decision regarding bonuses and promotions.

However it is possible for individual metrics to be contaminated by external factors, factors the engineer could have no control over. In that case the metric would not be an accurate representation of their work and would in fact be harmful to all parties through spreading misleading information.

2) **Apparentness:**

A good metric must also be apparent. If an engineer is unable to understand exactly why he is considered to be under-performing, they will find it difficult to improve since they do not know in what way they could better themselves. This understanding could derive from the metric being a number with a conceptually simple formula, or perhaps from it having a clear connection with how the programmer perceives their own work. If the programmer finds themselves in a situation where they are told to improve but do not know how, they may quickly lose motivation and seek alternative employment.

3) **Moderation:**

Finally even a good metric must be used carefully and thoughtfully if it is to do good. No matter how carefully thought through a metric is, focusing it one only a part of the engineering process invites exploitation by those measured, which can have a detrimental effect on the company. Furthermore metrics used to measure individual developers' performance should accommodate various positions they hold within their teams: some may be more likely to produce large volumes of code, while others may generally provide fewer lines of code but that code would be patching existing bugs.

It is clear that a number of metrics are required for a fair and accurate assessment of software engineering. Some of them may even encourage diametrically opposite development practices, and it is in this holistic approach that software engineering can achieve maximum efficiency.

Metrics

1) **Velocity**

Velocity a metric that measures the amount of value a team is adding, usually in the number of features implemented over a short period of time. Velocity is a good 'overview' metric as a manager can use it to assess the progress of a team in the development cycle to set realistic deadlines for the customers.

An interesting related figure is the acceleration – how the velocity changes over time. If velocity is very volatile it may be indicative of a bottleneck or other breakdown in the engineering process, whereas a stable velocity shows a team with a streamlined, efficient development cycle.

2) **Iteration flow**

Iteration flow is a metric that sums up ticket status over a long period of time. It records the amount of time a ticket spends in each state, from start to finish. It provides an apparent summary of the individual’s or team’s work and highlights bottlenecks in the engineering cycle. Due to the broad spectrum of data it encompasses, iteration flow metric can be used by the developers and managers alike to review the former’s progress and effectiveness, in terms of what work is being done and how consistent the developer is in delivering it.

3) **Pull requests quality**

A composite metric, it measures a ratio of accepted to rejected pull requests, and/or the the percentage of pull requests that fail to build or pass the existing tests. This metric focuses on the quality of the code, especially assessing its quality when added to an existing codebase.
While not explicitly useful to assess the quality of the code produced, these numbers go to quantify how effectively a team works. If a team’s pull requests quality has dropped, this could be symptomatic of all sorts of underlying issues, like individual developers under-performing or code reviews not been effective.

4) **Code churn**

Code churn is typically measured as the percentage of a developer’s own code representing an edit to their own recent work. It is measured by counting the lines of code that were modified, added and deleted over a short period of time such as a few weeks, divided by the total number of lines of code added.

If the code churn is high, a conclusion that seems apparent to make is that the quality of code the team produces is subpar and may show the team’s cohesion is low.

However it is very possible for a perfectly fine team to have a high code churn. For example in order to prototype quickly, like in developing a conceptually new feature or trying out a new framework, the engineers could have a high code churn. In that case it is completely justified to neglect the code churn metric, and this goes to show that any metric must be applied carefully not without human insight.

5) **Mean time between failures**

This metric assesses the quality of code that has found its way in production. It generally reflects the presence of serious bugs (P1 or P2). The stability and security of an application are arguably its most important characteristics, and the responsiveness of a team to issues in either of these categories links closely with the user experience, and through that to customer satisfaction.

Algorithmic Approaches:

Now that we have outlined the metrics we could be attempt to come up with a way to measure them consistently, accurately and ideally automatically.
One possible approach is to use machine learning. Metrics fit in nicely with machine learning as they are essentially a time series of effectiveness of software engineering. Regression analysis lends itself naturally for evaluating metrics.
It can highlight any anomalies in the historical data and essentially telling the manager exactly when the team failed to achieve. This narrows down the focus of the inquiry and thus simplifies location of the problem, be it team-wide or individual developer related. Given sufficient data such analysis could even be used to classify developers based on their performance trends, like matching individual developers' performance to that of past employees who had since grown to senior positions. Even though relying on this method alone is perhaps to narrow-minded it is an excellent way to inform HR decisions.

Regression analysis may also be used directly for inference. Team’s future work in may be predicted by a ML algorithm with a degree of accuracy depending on robustness of the algorithm as well as quality of training data. Such predictions help to estimate deadlines, suggest the aspects of software engineering that are likely to be neglected, and many other things, depending on the type of metric fed into the algorithm.

Computational platforms:

A more systematic approach to measuring software engineering metrics requires the use of one or more computational platforms. Instead of building a metric gathering framework entirely in-house, it is generally a more prudent approach to use one of the many readily available solutions.

Github API

Github is the largest software version control provider with some 37 million developers using the services of the platform. It was recently purchased by Microsoft. Github gathers and stores a large amount of statistics on most aspects of software engineering process. The data involving public repositories can be accessed via Github API and private repositories can also be accessed by supplying the necessary credentials.

Github API is very simple and fairly intuitive to use, user HTTP requests are receive a JSON response.
Depending on what the company wishes to measure the HTTP requests can be refined and be automated, to the point where developers’ progress is

constantly passively monitored; should any anomalies arise, team lead can be notified on short notice.

The main use of Github API is gathering analytics on the code itself, like lines of code added, modified, deleted, or the number of commits. However with these fairly basic statistics an analyst can calculate and infer further metrics, like using the lines count to calculate code churn.

One downside, albeit somewhat subjective, of using Github is the fact that these statistics come from an external source. Admittedly Github has a reputation for being a reliable service, but its recent acquisition by Microsoft could incentivise some developers to move their analytics frameworks elsewhere.

Bitbucket

Bitbucket is the second biggest provider in the software version control industry, with some 10 million software engineers using its services. It is owned by Atlassian and it is marketed as a alternative to Github for developer with private proprietary code.

As such one of its most interesting features is that it allows its users to set up their own local Bitbucket servers. This allows their personal and repositories related information to be stored locally and thus eliminate the need to rely on a Microsoft owned product.

Another of its features is the close integration Bitbucket has with other Atlassian solutions, like Jira, Bamboo and others. This integration simplifies the automatic extraction of metric data from raw commits developers make, providing a common framework an entire company may use for its internal purposes.

Testing suites

Testing suites are an umbrella term used to describe a collection of tests that validate various scenarios that compose the application. Unit tests validate internal modules' correctness, integration tests determine how well these modules are glued together, regression tests examine the effect of change in functionality of a module on the application as a whole. Most such functional tests are done automatically with little more than a mouse click and the outcomes of these tests is readily available to be monitored and analysed. Setting up a system to record these does not pose a major technical challenge. Such system could be tailored to monitor exactly what the company wishes. Moreover since all the data is stored and assessed internally there is no need to expose too much information about the internal software development practices.

Ethical Concerns:

Finally let us discuss the ethical concerns gathering and using metrics to measure software engineering presents.

Threats from without

In the last few years the concern that companies possess too much information about us has continued to grow. Legislation like GDPR has come into effect to prevents the firms from storing and selling their consumers' personal data. The amount of data stored on individuals should be minimal and the data stored should be stored for an explicit reason. This is designed to protect said consumers' identity in case of a security breach or should the firm wish to sell their emails and telephone numbers to the highest bidder. However with employees the situation becomes more difficult. Firstly companies are legally obliged to store some personal information on their employees, and thus they can find their personal information online should their employer's IT be compromised. Secondly the metrics calculated about individual developers will likely remain in the system for decades to come, so a today's developer in twenty years may find themselves questioned about something they did at an entry-level position decades prior. This lack of right to be forgotten is certainly a theme in 21st century, but this does not make it ever more ethical.

Threats from within

Many firms have a corporate culture that considers it perfectly acceptable to wonder what their employees do outside work hours. The answers to their question: How much time do you spend on open-source projects? What is your Stack Overflow reputation?, may well be used as metrics for internal assessment by the HR or the developers' supervisors. The argument is that doing work outside the workplace is an indicator of a good employee, that the things they do while working and resting are essentially the same. Personally, I find this entirely line of questioning unethical, as it is very much a breach of personal space and time, and the fact that it is not an uncommon occurrence in the industry is appalling. In pursuit of new metrics do measure just how good a developer someone a company may encourage unhealthy behaviours in their employees, from perpetual sleep deprivation to social deprivation.

Conclusion:

I believe metrics are a fantastic idea that allow developers to be assessed against their peers accurately and fairly. For leaders, metrics promote knowledge and understanding allowing them to make more informed choices for their employee's ultimate well-being. For developers metrics are a useful tool to assess their own progress and position within a team, enabling them to overcome their weakness and amplify their strengths. A healthy competitive environment, quantified by the correct metrics, could do wonders for workplace productivity, just as one based on poorly design metrics could do harm.

Metrics can be misused and must not completely relied upon. Ultimately metrics are just number and not matter how well thought through they might be, they still do not tell the whole story.
It is up to the person analysing the metrics to interpret them correctly, to draw the right conclusion by examining not just a metric, but perhaps a number of them, or listening to the developers in question.
Metrics can exploited, with a developer achieving on paper but still performing poorly in reality. It is for these reasons that it is imperative for the managers and the developers to understand the whats, hows and whys or metrics because if something is that important, it has to be done right.
Finally, careful ethical considerations must be kept in mind in basing any judgment on metrics, because of their personal implications permeating the life of a software engineer.

Vladimir Ustimenko
17306631

Bibliography

https://engineering-management.space/post/software-engineering-kpis/

https://www.linkedin.com/pulse/20140324073422-64875646-caution-when-kpis-turn-to-poison

https://blog.usenotion.com/8-essential-software-development-metrics-for-team-productivity-273737868960

https://www.nap.edu/read/11292/chapter/5

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

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

https://geekflare.com/choosing-ml-algorithms/

https://www.softwaretestinghelp.com/types-of-software-testing/