

## **Measuring Engineering – A Report**

The information age is well and truly upon us – where data is treated as a valuable commodity, and there are a multitude of businesses of massive wealth that exist, with their immense value coming almost purely from the value of the data that they generate.

With data being as valuable as it is, it's no wonder that we're now seeing data being collected, analysed and tracked more and more commonly in almost all aspects of life, in the world of business and leisure, in education and in entertainment, and so many more areas of life. This, of course, means that data analysis is becoming more prevalent than ever in the workplace, too. It's completely understandable – businesses are all about efficiency, and the best way to verify that a business is operating as efficiently as they can is to collect as much information as possible, and to use this information to see what the businesses strengths and weaknesses are with regards to efficiency.

Data, and the knowledge that we can gain from its analysis, has become essential for the growth, improvement and expansion of businesses in the modern era, and for this reason, there is a huge incentive for companies in the world of technology to perform the same data analysis on their employees. However, determining the efficiency of software engineers and their output is a task that has proven more difficult than doing the same for many other fields of work. This report will outline many of the suggested methods and approaches of measuring the efficiency and effectiveness of a software engineer, as well as many platforms from which this measurement can be done, and, finally, a number of ethical concerns linked to the collection of data and analysis of employee performance in this way.

The measurement of a software engineer's efficiency, and whether or not it's possible to actually do so, has been a rather controversial topic for some time, with many people believing that code quality is largely based on context, that you generally can't simply look at a segment of code and immediately recognise its efficiency or quality, as that can only be seen when looking at the software as a whole. In addition to this, many people believe that investment of a software engineer's time cannot inherently be gauged as good or bad, as there is potential for a large amount of time to be spent thinking instead of writing code, with this more careful approach then leading to better code in the end. In comparison, code that is written without much prior thought and consideration for structure and design may have more functionality achieved at a faster rate, but, the code is much more likely to be buggy, difficult to read, and harder to expand upon and add functionality to.

This dilemma creates the question of what pieces of data to track in order to determine how effective a software engineer is at completing their task. The immediate assumption may be to measure things like lines of code written by the engineer, or perhaps the number of bugs fixed or introduced by the individual. This approach is problematic, as it introduces a number of negative side-effects when put into practice, in that it rewards bad coding practices, and as a result, creates bad habits in the programmers under analysis. Measuring lines of code simply encourages software engineers to use more code to complete a task that can be done in less, which results in code that is likely harder to read and maintain, as it's nowhere near as concise as it could be.

As for rewarding bug fixes, this simply rewards introducing bugs into the code, specifically so that they can be fixed. Again, the result of this is lower-quality code, that takes longer to be completed, as time is thrown away by fixing bugs that have essentially been artificially introduced. Similarly, measuring data such as objectives/projects completed will only

reward software engineers that push out unrefined, bug-ridden code and projects that have been stripped down to a bare minimum of the specification, instead of releasing a complete, release-ready version of the project.

Yet another approach could be to measure the number of commits that are made, however, as with the previously mentioned approaches, this promotes bad practice, in that it rewards committing code at a much higher rate than is necessary. This, like measuring bug fixes, will lead to code taking more time to be completed, as programming time will be wasted on making frequent irrelevant changes, slowing down the turnover of projects.

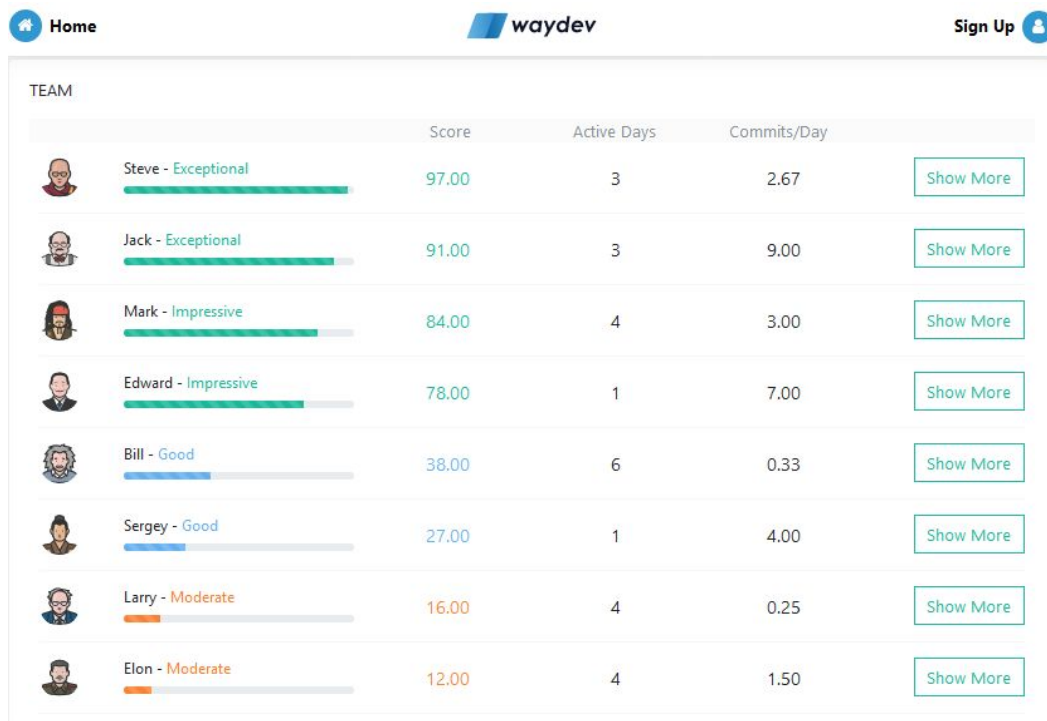
Previously, a purely algorithmic approach to measuring productivity was proposed and thought to be a solution to this issue. To try to account for the negative drawbacks of each of the metrics that I have described, these algorithms attach a weight to certain metrics based off of the overall problem the software looks to solve. For example, when measuring lines of code, as opposed to purely measuring raw numbers of lines of code written, this may be scored by using an equation such as:

$$\text{Score} = (\text{NL} + \text{RL} + \text{EL}) * \text{AAF}$$

Where NL = New lines, RL = Reused lines, EL = edited lines,  
and AAF (Adaptation Adjustment Factor) =  $(0.4 * (\% \text{ design edited}) + 0.3 * (\% \text{ of code edited}) + 0.3 * (\% \text{ of integration required vs integrating new code}))$

This equation, proposed by Dr. Richard D. Stutzke, although a step above raw, unprocessed measurement in an ideal scenario, can still fall victim to the same pitfalls, as it is still comprised of a number of factors that are open to manipulation in order to boost a perceived score, such as new lines of code for example. This does not prevent this manipulation from occurring, it only lessens the extent to which a single metric can warp the outcome of the analysis.

A service platform that uses metrics such as those previously mentioned to determine the effectiveness of software engineers is WayDev. This product is marketed to non-technical members of management, and tracks the activity of a team and its members.



The screenshot shows the WayDev interface with a 'TEAM' section. It displays a list of team members with their performance metrics. Each member has a profile picture, a name, a performance level (e.g., Exceptional, Impressive, Good, Moderate), a score, active days, commits per day, and a 'Show More' button. The scores are color-coded: green for Exceptional/Impressive, blue for Good, and orange for Moderate.

TEAM		Score	Active Days	Commits/Day	
	Steve - Exceptional	97.00	3	2.67	Show More
	Jack - Exceptional	91.00	3	9.00	Show More
	Mark - Impressive	84.00	4	3.00	Show More
	Edward - Impressive	78.00	1	7.00	Show More
	Bill - Good	38.00	6	0.33	Show More
	Sergey - Good	27.00	1	4.00	Show More
	Larry - Moderate	16.00	4	0.25	Show More
	Elon - Moderate	12.00	4	1.50	Show More

**A sample screenshot of the WayDev interface – Score is based on time spent working (Note: this doesn't mean that more individual days in which an employee is active is considered better, nor does more commits.)**

Although this isn't inherently measuring lines of code written or commits per software engineer, the end result is the same – the platform rewards spending more time than is needed to complete a task. As a result of this, WayDev has a generally poor rating amongst those that understand the difficulty of measuring the effectiveness of a programmer.

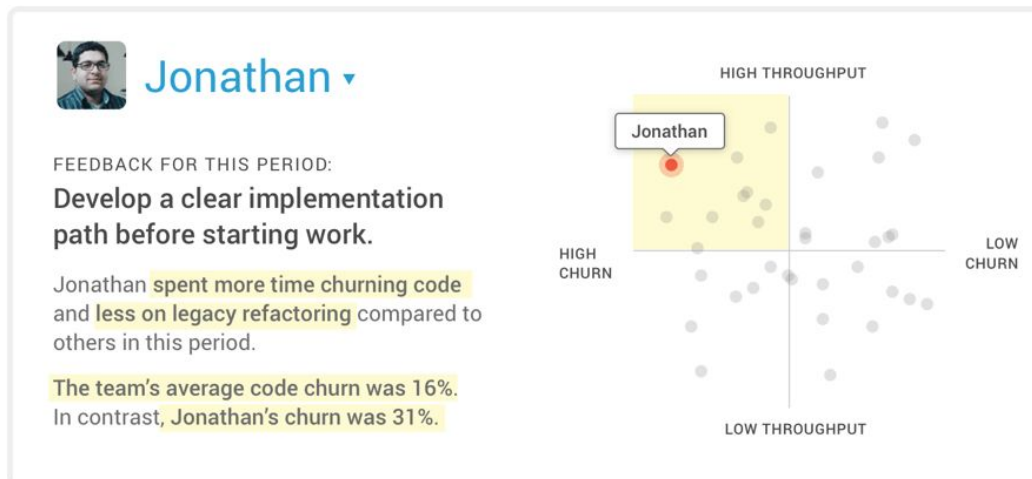
An approach that has seen more success and approval is that of a service such as GitPrime. GitPrime uses an algorithmic approach similar to that proposed by Stutzke, but also looks at the areas of code in which changes have been made, for example, if a programmer is adding code to areas that they aren't necessarily expected to be working on, i.e, helping others in the

team. This allows management to see how much time programmers spend working on what they've been specifically assigned to. On top of this, GitPrime also looks at how these statistics change over time, allowing a full view of how a team's productivity and focus adapts over the course of weeks and months, and in response to changes in direction, management or other factors which may influence productivity. Another statistic GitPrime tracks is code 'churn' – the amount of code that is rewritten and replaced in the project in a short space of time.

In a blog post, GitPrime themselves compare code churn to “...writing a postcard and then tearing it up and writing it again, and then again. Yes, you technically wrote three postcards, but in the end, only one was shipped so we're really talking about one postcard worth of 'accomplishment' from all that effort.” Tracking code churn is a useful way to track who is making high quality contributions to the project and getting it right the first time around, in code that sees itself remain in the project for long periods of time, without the need for bug-fixing, tweaking or maintenance.

This reveals to us the key that GitPrime uses to discover a more effective method of determining a software engineer's productivity – applying the context to the data. As programming, and its effectiveness, is so heavily context-based, it's important to gather as much information about the context of a situation as you do about the code itself in order to try to determine the value of a software engineer and their contribution. However, GitPrime is still an imperfect solution, as it is still unable to generate a full picture of what we're looking for in some scenarios. For example, a software engineer could commit a large amount of code that is essentially irrelevant to the completion of a goal in the project, but if it goes untouched as a result, the code, and it's programmer, could technically have a low churn rate, making it appear like a very solid contribution to the overall project. This creates a false positive for the software, promoting more bad behaviour, much like many of the previously mentioned approaches for

measurement, although, in this scenario, it's worth noting that GitPrime will also show the area of the code that the programmer is committing to, and in the case where a programmer is committing code to an area of the project that they haven't been assigned to work on, this activity will become obvious in the data that is displayed. Overall, it is certainly an improvement over the previously-discussed approaches, as it minimises the damage a single piece of manipulated data could do to an end result even further.



**GitPrime gives person-specific feedback based on collected data, and can show who may be the stronger or weaker members of a team.**



**GitPrime shows the daily progress of a developer across all repositories, breaking down progress into the different types of contribution.**

Many approaches have been suggested by those that have strong doubts that using data generated by code itself will ever be the solution to measuring the effectiveness of software engineers. One of such suggestions is to not look at code or project-based output, but to look at the software engineers themselves, and to measure their job satisfaction. This is due to the belief that things that get in the way of productivity are the main things that negatively impact a software engineer's job satisfaction. By this thinking, removing these barriers to productivity will make for more satisfied software engineers, increasing their productivity. Alongside this comes the idea that workers that are satisfied in their jobs are most likely to be enthusiastic and excited about their work. This enthusiasm will then prove to be a driving force behind maintaining high productivity. This has an added bonus of a satisfying working environment creating a positive reputation among software engineers for a business. A great reputation attracts great employees, which will create a strong workforce for the business. Dan Fabulich, Principal Engineer at Redfin, is a strong advocate for this approach to ensuring a productive software engineering team, and goes into great detail about many of his arguments for using this approach as opposed to more traditional data analysis in his article "You Can't Measure Software Engineering Productivity, so Measure Job Satisfaction Instead".

One of the things that stood out to me in this article was Fabulich's description of Redfin's approach to measuring the job satisfaction of its employees. Their entire analysis centres around a survey that employees fill out, saying how willing they would be to recommend Redfin to a friend, as well as explaining their positives and negatives about their job. To me, this doesn't seem like a perfect way to determine this, as it can be difficult for an individual to rate their satisfaction on a simple 1-10 scale. I believe that measuring satisfaction isn't always as simple as a short survey every now and then, as it can so frequently change, over both long and short periods of time. For example, a normally satisfied employee could be having a rare,

particularly bad day, for reasons unrelated to their work. If presented with a survey, they could then provide a response that is tainted by their current mood at that exact point in time, but doesn't fully reflect their real feelings about their job over the period of a number of months.

An alternative approach to the surveying method of trying to measure job satisfaction that has been suggested is to apply a knowledge of changes within the human body as a result of different levels of happiness to data that can be collected from employees. This suggestion stems from scientific research that suggests that there are certain patterns of activity in the human body that can be correlated with happiness or unhappiness, and tracking these patterns can help determine how an employee feel and reacts to a given situation in the workplace at any time. The proposed method of doing this data collection is to use wearable technology that each employee will have on at all times while at work. In terms of the actual information that can be collected and analysed, this may potentially be a more accurate method of discovering the genuine job satisfaction levels of employees within a company in a lot of cases – however, the ethical concerns surrounding this method of data collection are massive.

As a whole, I believe that it's totally fair to measure the efficiency of a software engineer and the team in which they work. Other employees have their efficiency tracked – salespeople have the amount that they sell analysed, delivery drivers have the time it takes for them to deliver a shipment analysed, why shouldn't a software engineer have *their* efficiency tracked? We're no better or worse than those in any other job, why should we be treated differently? The problem is, at this point in time, there's no known, accurate method of measuring a software engineer's efficiency that's as precise or trustworthy as there are for most other jobs, and some of the potentially more effective methods have massive ethical concerns surrounding them. With regards to tracking changes in bodily functions in order to gauge happiness, this seems to be completely ethically blind to me.



Not only is it a huge invasion of the privacy of each employee, this method also has questionable applicability to many potential members of the workforce. What would happen in a scenario where an employee has had a factor completely external to their job affect their happiness? If an employee is suffering from depression, or is in the midst of a crisis in non-working relationships or home life, will this impact their job satisfaction 'score'? If measuring software engineering productivity is difficult because of how context-based programming is, surely measuring a human's happiness is bound to be even more difficult because of how context-based life itself is. Personally, I believe that, looking past the privacy concerns completely, if this technology and approach were to be implemented, there would have to be significant statistical and scientific backup to prove that the technology will treat every potential subject fairly, and will be able to account for situations in which a non-work related incident is impacting the employee's happiness.

In conclusion, I believe that as software engineers, we are by no means entitled to go about our jobs without being subject to the same conditions as anyone else in any other job. Our performance, and its consistency, should be held to the same standard as anyone else's. For that reason, I personally have no issue with the *idea* of software engineers having their productivity tracked and analysed by an employer, in the same way that this is done for countless other jobs. However, in reality, measuring the productivity of a software engineer is immensely more challenging and abstract than doing so for other professions. Software Engineers are problem solvers, and when faced with the problem of figuring out how they can come across better to whatever system is tracking their performance, any and all flaws in the system will be exploited by many employees, which may only leave the business in a less efficient place than where it started, only now they'll have data, that may have cost them a large sum of money to gather, that tells them the opposite.

I think the solution to this problem is to go in the direction that services such as GitPrime have, only with much more data. Combining the more tangible data that can be collected based on a software engineer's work activity, along with the more abstract data that can be measured, such as job satisfaction and the employee's happiness in general, may potentially be the most reliable way to gauge the productivity of a software engineer in the workplace. Having more data to analyze means that the effect of data manipulation or a poor reading can be lessened, while having a number of pieces of information pointing in the same direction can allow for greater confidence in understanding how well a software engineer is performing. If this were able to be achieved, it would not only help cover the weaknesses that any one particular measurement may have, but it will also hopefully give much more consistent, accurate, and trustworthy results. Machine learning has seen a number of trials in analysing the value of an employee, however, at this point in time, it has proven wildly inconsistent, and even reportedly shown results that have been massively gender-biased, which shows that this technology is by no means ready for a large-scale implementation. That being said, as with many machine learning projects, given time (and much more data), this technology could prove to be a great tool in measuring software engineer productivity in the future.

In closing, I feel that it's fair to say that no perfect solution to this problem currently exists. In my opinion, the best thing to do is to find the least flawed solution, not necessarily a perfect one. On top of this, results that come out of these solutions should not be blindly trusted. There are ways in which a software engineer can be productive without writing code at all, and things like this must be taken into account when it comes to recognising which members of a team are stronger or weaker. However, a more rounded and solid solution may not be terribly far away, if our capabilities of collecting, analysing and applying context to data improve as they have over the past number of years. This may prove to be an exciting space to keep an eye on in the near future.

### References:

<https://redfin.engineering/measure-job-satisfaction-instead-of-software-engineering-productivity-418779ce3451>

<https://dev9.com/blog-posts/2015/1/the-myth-of-developer-productivity>

<https://app.waydev.co/demo>

<https://www.gitprime.com/product/>

<https://blog.gitprime.com/why-code-churn-matters/>

[http://www.hitachi.com/rev/pdf/2015/r2015\\_08\\_116.pdf](http://www.hitachi.com/rev/pdf/2015/r2015_08_116.pdf)

<https://www.scss.tcd.ie/Stephen.Barrett/teaching/CS3012/>

<https://www.bbc.com/news/technology-45809919>

<http://www.dtic.mil/dtic/tr/fulltext/u2/1014815.pdf>

[http://csse.usc.edu/afcaa/manual\\_draft/3.%20Collecting%20Metrics.pdf](http://csse.usc.edu/afcaa/manual_draft/3.%20Collecting%20Metrics.pdf)