## Intro - What are Software Engineering Metrics and why we need them.

Software engineering metrics are the way in which we measure the productivity or output capacity of software engineers. They provide a quantifiable measurement of any activity involved in software engineering. Fenton[[1]](#footnote-0) describes it as “*... measuring and predicting software project costs, measuring and improving productivity, and measuring and predicting the quality and complexity of software products*”. They enable us to see what aspects of the software engineering process are working at an efficient rate and what aspects require refinement to increase productivity and project turnaround.

A software metric in and of itself is a numerical value that is extracted from the software project. Metrics can be seen as either “product metrics or process metrics”[[2]](#footnote-1). Lim[[3]](#footnote-2) defines these as “*Product metrics are numerical values extracted from some document, or a piece of source code. Process metrics are numerical values that depict a software process such as the amount of time required to debug a module.*“ Software metrics can also be classified as result and predictor metrics. Predictor metrics are product metrics that can be used to predict the value of another metric. This is called the result metric.

The key question although is as to why we need them. Surely if we arrive at the finish line regardless, the process is irrelevant? This philosophy tends to plague the software engineering world as the process of developing tends to differ greatly from person to person but this is why we need metrics for. It enables us to ascertain what are they key aspects and habits that lead to good software development time and increases productivity as a whole and what habits lead to long development times and leaves projects and products in development hell. Ince summaries the use of software metrics as the following:

* As a means to predict the resource requirements for later parts of a software project, as requirements are always changing.
* As a quality-assurance enforcement mechanism.
* AS a mechanism for assessing the performance of staff on a software project.
* As a way to assess development methods and their potency.
* To be used as a way to assist development staff procure a quantitative estimate of the quality of their work.

By using software metrics we can prevent running overtime and poor productivity and also prevent running over budget and encourages good practices in software development.

## Obtaining Metrics

Obtaining metrics for software engineering tends to be a more difficult task than obtaining them in other fields. This is often due to the very nature of software engineering as a whole as the whole process of creating products and systems tends to be quite subjective to each engineers process. This leads to a lack of an objective way to quantify software engineers. Past attempts such as using Lines Of Code failed as they don’t account for a whole range of factors as will be discussed later.

The best way to get metrics is using subjective methods. In any organisation we can usually tell who the most productive developers are, not in an objective fashion such as how many lines of code they produce, but in a subjective fashion by looking at their habits which in turn we can quantify, giving us performance metrics we can use to compare and contrast engineers with.

They key areas on which we can judge a software engineer on is productivity,engagement, attention to quality, code base knowledge and management, learning and skills and personal responsibility[[4]](#footnote-3).

### Analysing the group vs the Individual

Software Engineering is rather unique compared to other roles. It’s a very solo role with a huge team focus. While this statement may seem like an oxymoron, writing the actual code is often done on an individual basis but all code that is written is used by the team and in fact the company as a whole.

This is why it’s necessary to both analyze the team as a whole but to also analyse the individual. We can analyse the team by looking at their overall output using metrics such as Cumulative flow diagrams and Velocity. These types of metrics enable us to look at the project as a whole as it sums up the effort made by all members of the team to create a metric for the team as a whole. But there lies a problem in this as we are unable to ascertain from this metric who is contributing what to the project. The majority of a project could be done by one person whilst their teammates drag their feet, which hinders the team's productivity as a whole.

To solve this problem, we also need to analyse each individual contributions. We can do this in numerous ways such as looking as the amount of lines of code committed although this metric is severely lacking. A better metric would be by doing code reviews for each individual on the team and from their we can assess on a subjective and personal level the output of an individual.

### Productivity

* Does the developer get a reasonable amount of features implemented in a certain time frame?
* How bug prone is their code and how are they are fixing them?

### Engagement

* How engaged is the developer on the given project?
* Do they communicate effectively with their team and clients?
* Is the developer committed to delivering on time?

### Attention To Quality

* Does the code work as intended?
* Is the code thoroughly tested before deployment?
* How many bugs get reported against their code?

### Code Base Knowledge and Management

* How well does the developer understand the code base?
* Adherence to coding guidelines and techniques
* Do code reviews reveal a minimum of problems and discrepancies?

### Learning and Skills

* Is the developer constantly learning and improving their skills?

### Personal Responsibility

* Does the developer assume fault lies with their code and take responsibility or not?
* Do they take responsibility to make sure their code works correctly and is tested correctly?

### 

### Past And Present Philosophies For Obtaining Metrics

While obtaining metrics for software engineering has always been a thing that has tried to be achieved in the industry, the approaches to getting these metrics has evolved and changed immensely over the years. There’s been a huge shift from just looking at the numbers objectively to evaluate performance to a more subjective case by case analysis factoring project size and team capabilities etc to judge performance.

#### LInes of Code Metric

One of the initial metrics considered was ‘Lines of Code’ or LoC. As there’s a correlation between the amount of code written and the productivity level of a developer, it could be said that more line of code equals better products.

This hard metric failed for a number of reasons. “LOC was simply a measure of how much you typed, and not how quality code you wrote. A strict use of LOC as a metric could introduce dysfunctional dynamics through people inflating their LOC by not refactoring their code properly and creating future maintenance problems”[[5]](#footnote-4). The amount of code you wrote doesn’t correlate to how much quality code you actually have. This metric could lead to developers beefing up their LoC count to skew the metric in their favour, creating unnecessarily convoluted code just to increase their line count.

#### Productivity measures

Productivity metrics are used to measure the productivity of personnel during different times in the software engineering process. Fenton[[6]](#footnote-5) proposes that productivity be measured as a function of value and cost than size of output divided by effort.

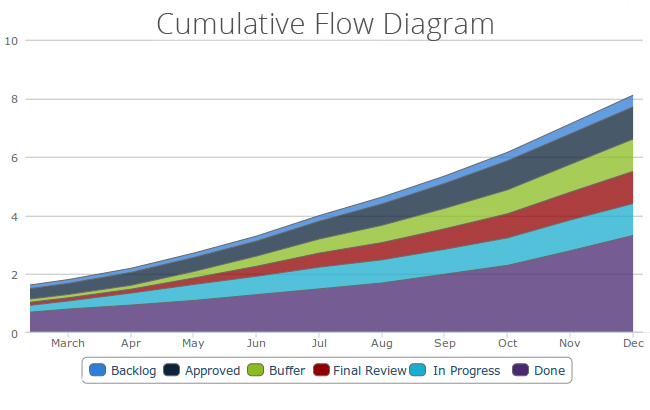
In general people do not like to be monitored and measured. If a developer knows that they are being monitored, they may attempt to skew the results in a way that may be unrepresentative of the general case. Due to this, “productivity should be viewed as an attribute of the human resource[[7]](#footnote-6)”.

Productivity can be increased in 2 major ways:

* By using more sophisticated toolkits that can achieve more with less
* By improving the management of developers, helping them to stay on schedule and deliver the product on time.

#### Cumulative Flow Metric

Cumulative flow diagrams were introduced in 2003 as an alternative to bum up charts. CFD’s represent the amount of work done in a given time frame.



Source: <https://kanbantool.com/cumulative-flow-diagram>

Above is an example of a cumulative flow diagram. In this figure, the quantity of work in each iteration is shown in predefined stages such as backlog,approved,buffer,final review and done.

Using CFDs is not only useful for depicting work in progress for each section of the project but also to increase throughput and reduce lead times. By this we can estimate in advance the delivery date and lead times of projects. Another benefit of CFDs is we can monitor bottlenecks in development workflow and achieve continuous improvement by eliminating bottlenecks by analysing the CFD data. We can also get a sense of the scope of a project by looking at the CFD, enabling us to look at other metrics gathered in correct context.

“To sum up, the cumulative flow diagram enables teams to measure how efficiently they are delivering valuable, working product to the customer, and indicates where they need to focus their process improvement efforts.[[8]](#footnote-7)”

## Software Engineering Metrics Tools

While it’s nice to be able to talk about possible methods for obtaining metrics for software engineering, we should look and analyse real world tools that are on the marker to evaluate their worth to companies and if the insights they provide are worth the time and effort.

### Gitprime

“GitPrime is an organizational tool, pioneering a different way of measuring and communicating about productivity in software engineering.”[[9]](#footnote-8) This excerpt from their website tells us the company focuses on obtaining metrics from software engineers and communicating this data back to the company asking for these metrics.

GitPrime aims to be able to answer questions normally asked and answered on a subjective basis, such as :

* What was delivered previously
* How much time went to refactoring old code
* What bottlenecks are in the system
* How to identify struggling team mates and how to help them

They attempt to answer all these questions algorithmically by looking at just commits. Their algorithm works of the following things[[10]](#footnote-9):

* **Impact:** GitPrime also measures the difficulty of a change to the codebase. According to their docs, their algorithm is based on insertion points, lines of code, and number of files change.
* **Churn:** This metric measures how much code has been reworked. This number is affected when any contributor refactors any chunk of code and can represent wasted effort.
* **TT100 Raw:** This is the time it takes any engineer to produce 100 lines of code, regardless of the quality of that code.
* **TT100 Productive:** The time it takes for an engineer to produce 100 lines of code, after churn. This metric represents speed and quality, and GitPrime recommends that managers look at this metric together with TT100 Raw.

Gitprime also provides a host of other services such as work logs to see what developers have been up to recently and also spotchecks, which enables you to see how a developers performance has changed in the last month and spot abnormalities.

### Velocity

Velocity is another tool for obtaining metrics created by Code Climate. They focus on a more team orientated approach compared to GitPrime. They take into account things such as code reviews to generate insights as both a high level and a much more personal level.

Their algorithm is rather different to GitPrime as they factor in a host of different things not taken into account as much by GitPrime such as:[[11]](#footnote-10)

* **Pull Request Activity Level:** This metric estimates how much of your team’s attention a particular pull request is taking up. The number of comments, review cycles, and contributors influences how “active” a PR is. A dashboard that displays PRs by activity level gives a manager insight into which work is most “at risk” of holding up the team.
* **Review Cycles**: This metric calculates the number of times a PR goes back and forth between the contributor and the reviewer. The higher this count, the more problematic or complex a PR may be.

There are also other metrics similar to GitPrime such as :

* **Impact:** This metric reports the estimated difficulty of a change to the codebase, and thus, the impact a given change has on a project. Three variables are factored into this metric: the location of the change, the size, and the nature (e.g., is it a simple re-name of a variable or is it a more complex change).
* **Rework:** This helps illuminate the quality of the code that your team is producing. This metric is affected when a contributor reworks code that they, themselves have written. High rework can indicate that an engineer is stuck– possibly due to lack of familiarity with the codebase or changing requirements.

As for tracking progress on a product, Velocity has a host of features designed around teams such as Overview, which summarises the progress your team has made over time based on metrics such as impact, Pull request activity and others. Another tool they have is targets, which allows for teams to set targets based on previous metrics such as Impact and Push Volume.

In terms of actual personal level metrics, Velocity falls short in comparison to GitPrime. While they do have the tools to show this, such as Activity Log and Work in Progress, in comparison to GitPrime, their tools do not provide enough insight into who on the team in particular may be struggling or where the bottleneck in production may be at that particular moment in time.

In conclusion, for a more team orientated approach, Velocity has the superior tools as we can see how the team is doing as a whole whilst also getting some insight to the individual developers and it also takes into account more collaborative things such as Code Reviews, but GitPrime allows us to see on an individual level how each developer is doing, if they’re stuck and how best to help them improve productivity by assigning extra resources where necessary.

## Ethics of Metrics

When we obtain metrics in software engineering we need to consider the ethical repercussions of such metrics. While metrics enable us to quantify software engineering, they often don’t tell the full story.

No 2 people write code the same way. Using metrics to quantify things such as software quality can lead to some ethical problems. If 2 people write the same code but in different ways that ultimately lead to the same result, it’s very much up to the bias of the person who wrote the metric that determines software quality as to who’s code is better than the other. Here we need to be careful about the repercussions of the metrics we create as if we do not take care to measure things in a subjective manner rather than objective we can get caught with bias in our metrics which ultimate skews future projects for better or worse depending on where the bias lies.

Another ethical issue we need to take into account is consent to the data being collected on them. Members of a team in a software company need to know if they’re being monitored and explicitly consent to this to avoid unfairness. Take the following example:

“Team 1 consists of 4 staff members, 3 developers and one team lead. The team lead collects data on all developers to see how productive they’ve been over the last week. By the end of the week they have a staff meeting to discuss performance. Here they see a chart with their performance over the last week, with each member being ranked from most productive to least productive. The team leader then holds individual meetings with each about their performance over the last week and how to improve productivity.”

What’s the problem with this above statement? Well the 3 developers are never told and never explicitly agree to having their data being collected and metrics formed from this data. As far as they know it’s business as usual and everything is as normal. It’s unfair to monitor people if they never agreed to being monitored as this creates a sense of unease within the team and in fact tends to lead to less productivity as a result as focus gets taken away from the project and towards the fact of being monitored.

Another major issue is the metric data is shared with the whole team. A member of the team may not feel comfortable with their data being stored and shared. Especially if it paints them in a bad light.

The final issue with this is how the metrics are evaluated. There’s no accounting for external circumstances or the difficulty of task being taken on by each developer. In essence, the metrics are too objective which could lead to bad feedback in the individual meetings as a result. The lowest ranking developer may have had the most difficult problem to solve. They may have been ill for a few days and unable to work. We need to be careful with how we collect data and the interpretation of results as we can’t control all stimuli in an environment to be able to perfectly assess individuals.

#### Real world examples

Amazon, one of the world’s biggest software companies has an incredibly high staff turnover rate, with staff median tenure at just 1 year[[12]](#footnote-11). Why is this?

Amazon is notorious in the industry for having one of the most stressful workplaces as excellence is required constantly. This leads to the close monitoring of individuals and performance being monitored closely at all times. And everyone knows people don’t like to be monitored.

The constant monitoring and need to hit deadlines on time everytime can lead to a toxic work culture and high stress rate in employees which in turn creates unhappiness in employees making them leave after a short amount of time. This also tends to screw the work life balance which is very important as people need to switch off from time to time.

This constant need for excellence can also be detrimental. If an employee is underperforming for a period of time they tend to be fired. “Amazon is exceptional at removing its underperformers quickly. What enables Amazon to “fire well” is its strong culture, which [obsesses over results](https://amazonbound.today/blog/20474/output-matters-more).”[[13]](#footnote-12). While this is a good thing for the company, we need to take it on a case by case basis. We need to be subjective when assessing why they’re underperforming. Is it due to a lack of support? Personal reasons? Or are they truly not up to the bar? It’s morally unfair to fire based purely on metrics when extra factors that cannot be controlled may be part of the equation also.

## Conclusion

In conclusion, to obtain metrics for software engineering, we need to first determine who we or what it is we’re collecting data to obtain a metric on. If it’s an individual, we need to be subjective and take into an account extraneous factors that the metrics may not account for. For a team we need to be less subjective but we also need to be able to isolate weak links in the team and allocate resources as necessary based on the metrics.

As for how we evaluate our metrics, we need to take into account how we obtained the data and what it actually represents. Our metrics need to give us valuable insight into the development process and also into the behaviours of the members of the team. If they just give us arbitrary numbers that are meaningless without extra data, there’s no point in making the effort to obtain the metrics in the first place.

There are a number of tools available in the real world to obtain metrics based on what we’re attempting to learn: whether it’s about an individual or a team. Different tools take different approaches to algorithm design based off what we’re quantifying and as a result, what data points they use can differ hugely also.

The ethics of software engineering play a huge rule in how we collect our data and what we do with our data. We have to make sure who we collect data on knows we are collecting data and consents to us collecting and using the data. More important than anything else is what decisions we make based on these metrics. More often than not metrics don’t tell the whole story of software engineering process so we should use them with care and also take into account individual reports from people who worked on the software, lest we fall into the trap of assuming that an unfavourable metric means a weak team, which may be very far from the truth.

1. Fenton, N. E. (1991). Software Metrics: A rigorous approach. Norwich: Page Bros Ltd. [↑](#footnote-ref-0)
2. Ince, D. ( 1990, May). Software Metrics : Introduction. Information And Software Technology, 32 (4), 297 · 303. [↑](#footnote-ref-1)
3. Lim, C . E. (1994) Software Metrics For Monitoring Software Engineering Projects [↑](#footnote-ref-2)
4. Can Developer Productivity be Measured - https://dev.to/nickhodges/can-developer-productivity-be-measured-1npo [↑](#footnote-ref-3)
5. https://dzone.com/articles/lines-code-bad-metric-either [↑](#footnote-ref-4)
6. Fenton, N. E. (1991). Software Metrics: A rigorous approach. Norwich: Page Bros Ltd. [↑](#footnote-ref-5)
7. Lim, C . E. (1994) Software Metrics For Monitoring Software Engineering Projects [↑](#footnote-ref-6)
8. <https://www.researchgate.net/profile/Hazura_Zulzalil/publication/235009023_On_the_Current_Measurement_Practices_in_Agile_Software_Development/links/0deec52799054e8cfe000000/On-the-Current-Measurement-Practices-in-Agile-Software-Development.pdf?origin=publication_detail> (pg 5) [↑](#footnote-ref-7)
9. What is GitPrime Exactly - https://help.gitprime.com/general/what-is-gitprime-exactly [↑](#footnote-ref-8)
10. How to choose engineering intelligence tool - https://codeclimate.com/blog/velocity-vs-gitprime/ [↑](#footnote-ref-9)
11. Rezvina S. - How to choose an engineering intelligence tool - https://codeclimate.com/blog/velocity-vs-gitprime/ [↑](#footnote-ref-10)
12. Tim Johnson - The Real Problem with Tech Professionals: High Turnover - https://www.forbes.com/sites/forbesbusinessdevelopmentcouncil/2018/06/29/the-real-problem-with-tech-professionals-high-turnover/#3b314f694201 [↑](#footnote-ref-11)
13. Nick Dmitrov - Why Amazon’s Attrition is Higher than Attrition at Other Tech Companies -

    https://medium.com/@amazonbound/why-amazons-attrition-is-higher-than-attrition-at-other-tech-companies-b0de9cb2c4fe [↑](#footnote-ref-12)