Measuring Software Engineering

**By:** Bair Buyrchiyev (18319035)

**Contents:**

**Brief History…………………………………………………………………… 2**

**Why Measure? ………………………………………………………………… 2-3**

**Ways to Measure……………………………………………………………… 3-4**

**Determining Good Metrics………………………………………………….. 5-6**

**Computational Platforms …………………………………………………… 6-7**

**Ethics Concern………………………………………………………………... 7**

**Conclusion…………………………………………………………………….. 8**

**References……………………………………………………………….. …… 8**

**Brief History:**

In this modern and very connected age, software engineering has evolved to be something that countless companies employ within their computing department and that seemingly many people aim to work in, although this has not always been the case.

The term “*software engineering*” was coined by in the mid 1950’s at MIT where it was starting to form its own as a profession. Two NATO conferences on software engineering gave the discipline a much-needed popularity boost and officially marked the start of the profession.

The following two decades, 1965 – 1985 gives us a great example as to why measuring software engineering is very important to the stability of an ongoing project. This 20-year time frame is regarded as “*The Software Crisis*” due to how many projects ran over schedule and budget, were low quality, inefficient and some were never even delivered. There was a huge lack of programmers working together, leading to a massive workload being placed at the hands of a single person as well as having no communication between the engineers and the customer leading to poor specifications.

This era of software engineering were the mistakes that the profession improved upon drastically coming up to the age of the internet, giving us what we have as standard nowadays.

**Why Measure?:**

To prevent the example given above, the measurement of software engineering is employed to let companies get an understanding as to how productive a team for a project is, but don’t take this the wrong way, as if they only cared for productivity then they would never end up with a meaningful product suited to the customer’s vision.

This data can give important insights to what a project could be lacking in, be it quality, productivity, cost, or schedule and so then everyone working on the project can focus their efforts on improving vital parts of it. Improvement is one of the biggest reasons for measuring software engineering and is why many companies employ a multitude of metrics across a project.

What a company’s main interest is, pretty much always is a profit, so these metrics can be used to help get a better return on investment on the project, rather than spill out money through wasted resources. A software development manager is always looking out for their team, while also keeping profits in mind, these metrics can help them make better decisions as to not make any of their team stressed out or have to work overtime by identifying areas of improvement so the team works at a level pace throughout the whole project.

But, even with all these upsides to measuring this data, it can also bring up ethical concerns on singling out a single person for not providing enough. This topic will be brough up later in the ethics section.

**Ways to Measure:**

There are countless ways to measure software engineering, but currently the best way is through “*software metrics*”, which can mean a lot of things to a lot of people but it can be best described as the measurement of the software development process and its resulting product. Metrics are vast and encompass the whole array of ways, although there is a vast separation between “*good*” metrics and “*bad*” metrics.

Some of the most common metrics seen in the workplace are:

* Number of lines of code
* Code coverage
* Commits
* Development time

These are incredibly simple pieces of information to measure and gather, but can be considered not incredibly useful as it does not give a good insight as to how well the project is coming along, and is also quite subjective to the person that they are gathering it from.

An easy example of this is the “*lines of code*” metric which can easily be misleading because, there could be a huge discrepancy between a more amateur programmer and an experienced one, as at first glance you might be lead to believe that the less experienced programmer is more productive due to the fact that they may have more lines of code, but in reality, what a newer programmer can do in four lines of code, the experienced one will most likely be able to do in one.

Rather, a better way to measure a team’s performance is through the results of multiple metrics, giving a bigger picture to the manager as to make better decisions down the line. Production metrics are an example of this, combining metrics such as “lines of code”, “active days” and “level of effort estimate” tells the manager how much production a single person can output at a time, although this metric doesn’t always equate to progress as development itself is a non-linear activity and how estimation is only an estimation at best.

One of the ways that trends are normally tracked are through numbers, which on paper are the easiest and most efficient way to track metrics and are the best way to track metrics in the short term, but you must remember that software development is a long and time-consuming process and to make the best use of this time is to work on or improve parts that are based off metric trends. Trends make managers and workers come up with better questions on how to improve a part of software rather than pointing out the gap between a set of numbers.

Another good way to make sure that metrics are not just an incredibly high goal that managers set at the start of the project is by having multiple smaller goals throughout the working of it. A trap many organisations fall for is by setting a metric at the beginning to last throughout the life of the project and then when the project ends and the goal has not been met, criticisms such as, “you had a full year to reach the goal and you missed it” do not help any team members evaluate how bad or well they did but rather just take blame for no discernible reason, smaller goals set every week or so give the teams something more substantial to work on, it also gives managers better insight as to what the team is achieving in the span of a week, letting them know exactly how well they are faring with the objective.

This issue does not happen with all metrics however, as the more complex they get, the easier it is to tell how smooth a project is running, but with the complexity of a metric comes having a harder time being able to track said metric, as it could be something less tangible.

**Determining Good Metrics:**

Metrics need to have good purpose and reason to exist, as otherwise they are completely useless and serve no function other than to waste resources on worthless data.

There exist many types of metric that measure a whole plethora of items, starting from goal metrics which give the programmers a goal to work towards

A type of metric that is vital to a project is customer need, as if you do not fulfil the needs of the customer, then there is no reason to work on the project. This metric stands above all and is the best way to determine the efficiency of work if needs are being worked on. Past making the project to the customer’s specifications, making sure that customer satisfaction stays high is incredibly important and so managers make sure to make metrics based off what the customer’s reaction to the product is.

Making sure a program can run, while still being quick and efficient definitely has to be a very important metric as it would make no sense to deliver a program that does not even run or even compile to the customer. A multitude of smaller metrics can be used to fulfil this big one, such as commits that compile, code coverage and test code. These can be very good for catching edge case scenarios or just general bugs within the code. Having all these metrics together form a cohesive metric that covers the whole spectrum of viable and durable code.

Project management has consistently been one of the most useful parts of software engineering, letting a group work as one and co-ordinate without getting muddled up within any size project. Agile and Waterfall are the most widely used and popular and both have metrics to evaluate the effectiveness of how well the team is performing, be it either they finish early consistently, meaning that there is a lack of work coming in each sprint, or consistent missed sprint deadlines meaning that the workload is too much for the group to handle. This metric provides key data and a good reference on how well or poorly the team is performing on the project.

An important quality that metrics must have is that they are repeatable, meaning that the information they hold stays accurate and consistent even when measured at a different time, otherwise the metrics will become unpredictable and unreliable. If the metrics are repeatable and consistent then it is much easier to notice the values of the metric go up or down, possibly leading to something such as a higher production or the opposite. Along with this, metrics should also be language independent, meaning that no matter which language the team will be programming in, the metrics will stay relevant across all of them. These metrics should be broad and not focused in on a single language, because if so then say if they a new customer wants to have their project written in a different language, then the language specific metric won’t be applicable to the new project, leading to having to create new metrics which will waste time and money for the company.

Although metrics can setting targets easier, it can lead managers into a false sense of security as they may be too focused on specific metrics and miss out on other important metrics that could lead the whole team down a destructive path, an example of this could be when managers ask when a certain feature will be completed, which most programmers would respond when the coding of it is finished rather than when testing and bug fixes are started or yet to be finished. It is very important to judge metrics in a non-linear way and seek ways that the data can be used in the future to further improve the code.

**Computational Platforms:**

Now, for these metrics to be of any use at all, they need to exist within a computational platform that can calculate said metrics.

Git is the absolute most popular type of version control used across the entire world. The reason it remains so widely used is due to the limitless options it provides to source control. It is incredible easy to work on separate branches and create new branches which is probably the most useful aspects to Git along with merging which it handles incredibly well. Although Git may not be as easy to use as SVN, it remains much more powerful at the cost of a slight learning curve, as well as not having to be always connected to the main branch.

GitHub is the hosting platform for Git, it provides cloud storage to the code that developers write as well as providing an incredible number of tools to them. As Git remains restricted to the terminal, GitHub hosts everything on their website giving all your branches, history of commits, merges, and deletions an intuitive graphical interface that anyone can use. GitHub also provides ownership of code, letting managers give certain people access to certain parts of the code. GitHub also provides functionality in many programs that are used widely across many development groups, such as Eclipse and Visual Studio.

Gitlab is a platform that gives users the full power of Git, while also having many enterprise features such as authentication levels and continuous integration for an increased workflow.

**Ethics Concern:**

For as much benefit that measuring software engineering can give to the company, be it profit-wise, efficiency or time-wise, there are many ways that this data can be used for malicious purposes in order to punish a poorly performing engineer.

There are about a billion factors that a single person could be performing poorly. Hard and set rules for these things cannot exist for people because we are people. Imposing hard rules would lead to a much less efficient workplace as maybe the more slow and methodical people might be forced to meet a certain criterion every day to avoid losing their jobs.

In a world full of data and information floating around, it is hard to not assign people as blank faces only filled with the data they output, but money is a massive driving force for many of these businesses which may lead them to kick out lower performing workers in order to visualise a bigger profit margin.

There is no privacy anymore. Everything that is done on a digital device nowadays keeps a footprint, and your personal life should not spill over and affect your professional business life by having higher ups assess your performance based off external factors.

**Conclusion:**

Software engineering can be an incredibly useful for a team to finish a project to the specifications of the customer. But with this teamwork can come disorganisation, making way for the need for measuring it, this is where metrics come in and help try and solve efficiency and fix some issues in productivity.

As helpful as these metrics could seem, it is important that they are used for beneficial and non-malicious ways as to single a worker out. The statistics don’t lie but you must remember that they do not tell the whole story, as we are working with people here and not machines, there is no one hundred percent accurate way to measure a human’s activity.

**References:**

<https://en.wikipedia.org/wiki/Software_engineering>

<https://en.wikipedia.org/wiki/History_of_software_engineering>

<https://www.ukessays.com/essays/information-technology/software-crisis-of-the-20th-century-information-technology-essay.php>

<https://www.cs.purdue.edu/homes/cs307/slides/lecture14_2up.pdf>

<https://stackify.com/track-software-metrics/>

<https://martinfowler.com/articles/useOfMetrics.html>