# MEASURING SOFTWARE ENGINEERING: TIPS FRAMEWORK

## Contents

# Executive Summary

**What is Software Engineering?** Software Engineering is the application of a systematic, disciplined and quantifiable approach to the development, operation and maintenance of software, as defined by IEEE. Or in the words of the German computer scientist Friedrich Bauer, “software engineering is the establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines.”[1]

**Who is involved in Software Engineering and with which goals?** The current worldwide population of professional developers is estimated to be 23.9 million and is projected to grow to 28.7 million in 2024.[3] As of January 2020, GitHub reports having over 40 million users. These users engage in software development for different reasons, the most common being as part of their job in large organisations (both technical and non-technical), in startups or for personal projects. These users all have different goals for their software development efforts, but they can be brought to a common ground of three goals:

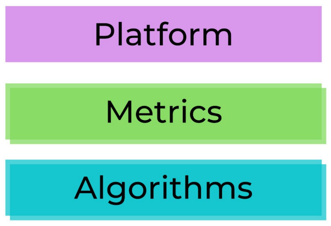
- Meet specific requirements
- High software quality
- On-time delivery

**Why is measuring Software Engineering important?** In order to obtain these goals, organizations and individuals find measuring their performance using metrics a useful way to consciously improve and increase said performance over time. A report published by Harris Analytics and Stripe shows that developers if used effectively, have the collective potential to raise global GDP by $3 trillion over the next 8 years.[2] It is therefore in the interest of the engineers themselves and companies to measure software development, ensure productivity maximization, reduce costs and improve efficiency.

**The Measuring Software Engineering Stack:**
So far we have established the goals of software engineering and the importance of measuring it, but how can we do so? There are academic reports dating back to 1999 attempting to address this question and large organisations like Google and Facebook, whose success is dependent on Software Engineering, still don’t have an ultimate formula to measure SE, but we do have many metrics and proposed hypotheses.

The picture below is the Measuring Software Engineering stack. It breaks down the components that make measuring SE possible.

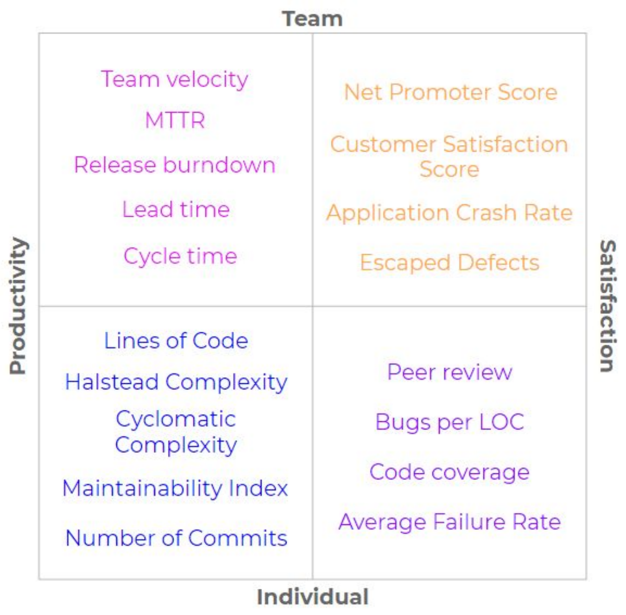
- There are companies offering tools and **platforms** to allow engineers to visualise their performance.
- These platforms draw their conclusions on productivity and efficiency from quantifiable **metrics**
- which are derived from **algorithms**.



The next three sections of this report will dive deeper into each component of this stack.

## How can we measure Software Engineering?

To keep the answer short, using **software metrics**. Software metrics are quantitative measurements of a software product or project, which can help management understand software performance, quality, or the productivity and efficiency of software teams. There are many different categories of software metrics: productivity, process, quality, reliability, etc but I have created the **TIPS framework** to capture and classify metrics that measure both individual and team productivity and satisfaction. The TIPS (Team, Individual, Productivity, Satisfaction) matrix below offers a holistic view of software engineering performance.



**Individual Productivity** These metrics measure the ability of software engineers to be productive, and therefore achieve on-time delivery.

**Lines of Code** is a formal and size-oriented metric which measures the lines of code written. This over-simplistic metric can be easily optimised for by engineers, encouraging copy-pasting code or unnecessary comments and discouraging refactoring. In addition, as different engineers use different coding languages which have different design patterns, comparing performance becomes difficult.

**Maintainability Index** measures how maintainable (easy to support and change) the source code is. The maintainability index is calculated as a factored formula consisting of SLOC (Source Lines Of Code), Cyclomatic Complexity and Halstead volume.[5] It is used in several automated software metric tools, including the Microsoft Visual Studio 2010 development environment.

**Number of Commits** is a measure of the time spent by an engineer on their code. It is an effective measure of activity levels but has flaws as an indicator of productivity because a large number of commits does not coincide with better performance, as they could be empty commits or including low-quality code.

**Team Productivity** These metrics measure the ability of the team as a whole to achieve on-time delivery and customer satisfaction. As teams have shifted to work following the Agile framework, Agile metrics have proven to be effective, and therefore outcome metrics are used to measure team productivity.[4]

**Team velocity** is an Agile metric that measures the number of story points completed by the team in the previous sprints. A story point is a measurement of how difficult it is to create a particular piece of work that has been assigned to a team. Sprints are periods of time with a set start and end date (typically two weeks) in which, ideally, work has to be started and finished. Velocity helps to understand how much value the team is providing to customers in a given time period and can be useful for teams to plan their future sprints and how much work they can expect to complete.

**Release burndown** chart can help teams understand how development is progressing, how much of the planned software functionality remains to be done, and when they can expect the release to be completed.

**MTTR** is short for Mean Time To Recovery. It is a non-agile indirect metric of code reliability. Indirect means that MTTR has been derived from fundamental code data. MTTR is the average time it takes to recover from a product or system failure. This includes the full time of the outage—from the time the system or product fails to the time that it becomes fully operational again.[6] Developers spend approximately four hours a week on “bad code,” which equates to nearly $85 billion worldwide in opportunity cost lost annually. This figure shows the importance of having a low MTTR.

**Individual Satisfaction** The metrics within this category focus on measuring the quality of the code delivered.

**Bugs per line of code** is a measure of code quality. NASA achieved zero bugs for the Space Shuttle Software but at a cost of thousands of dollars per line of code. Every individual or company has to decide a threshold that ensures enough code quality at a sustainable cost.

**Code coverage** is a measure of how much source code of a program is executed when a particular test suite runs. It would be optimal to measure test coverage for all types of tests: unit, integration, UI automation, manual tests and end-to-end acceptance tests as they can reveal quality gaps. Same as the metric above, a tradeoff between the quality of code coverage and its cost exists. For example, Google has spent more than a decade refining its coverage infrastructure and implementing and validating different academic approaches. Google’s infrastructure supports seven programming languages and scales to a codebase of one billion lines of code that receives tens of thousands commits per day.[7]

**Peer review** is a measure used by tech companies such as Facebook, Google and Uber. Every quarter co-workers have to review each other's performance and measure it on a 1 to 5 scale: 3 meets all expectations, 5 being exceptional, and 1 being unsatisfactory. Peer review is an indirect metric and can be very subjective.

**Team Satisfaction** With software deliverables being more-and-more customer-centric, customer satisfaction has become an important measure of software performance.

**Application Crash Rate** is the number of times an application fails divided by the number of times it was used. This measure is especially important for app development where the ACR considered acceptable has decreased over time as development practices have improved. A high ACR can lead to a high churn rate, which is the percentage of users who uninstall or stop engaging with an app over time, which is undesirable for software engineering.

**Escaped defects** is a metric of technical user satisfaction as it checks the quality as perceived by the end-user. It checks, per release or product component, how many bugs or issues were identified after the software was already in production. If it's increasing, it is a sign of a faulty quality process.

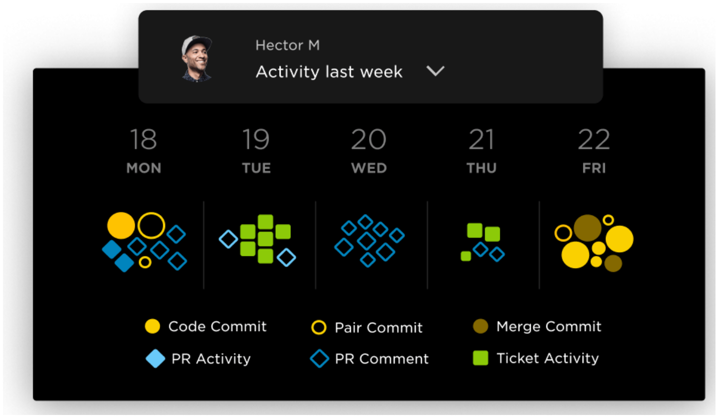
**Net Promoter Score** is the most popular measure for customer satisfaction. It's a number ranging from -100 (indicating no customers refer you to others) to +100 (all customers likely to refer you to others). The data for this metric is gathered from surveys of users (managers, clients or end-users) using the product delivered. It is calculated by subtracting the percentage of survey responses with a score between 9-10 from the percentage of survey responses with a score between 0-6.

## Which tools can we use to measure SE?

Now that we have studied the different ways in which individuals and teams can be measured, let's pivot to explore the different **developer productivity tools** available to allow engineers, managers and clients to visualise the performance of their software engineering efforts. The rising interest in measuring software over the past years has led to the creation of a competitive market for companies that can efficiently and accurately calculate insightful and relevant metrics. Here is an overview of some of these tools:

**Pluralsight Flow** aggregates historical git data using their API into easy-to-understand insights and reports to help make engineer teams more successful. During an interview, the CEO of Pluralsight Flow stated "In our experience, we've found the following five developer metrics are essential for all software managers," listing lead time, churn, impact, active days, and efficiency. Money quote: "We suggest focusing on these particular metrics because you can't track everything, and not every measurement is a key metric." [8]

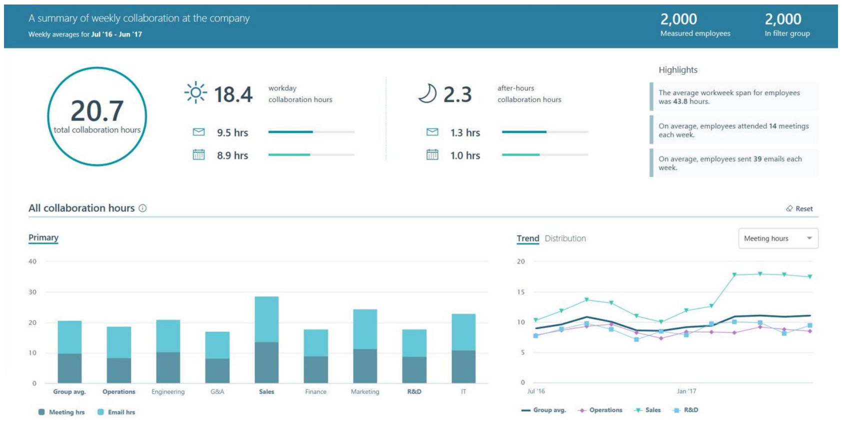
The diagram above shows the dashboard available to team members showing the number of different types of commits they have done, the number of tickets they have resolved and their PR activity.



**JIRA Software** is a tracking tool owned by Atlassian, an Australian software company that develops products for software developers and project managers. JIRA is an agile planning suite, or in their own words "to plan, track, and release great software." Jira software is popular in the Agile world due to its well-managed workflow mapping and issue tracking ability. To support Agile development cycle it has Scrum and Kanban boards along with various reports. They offer visualizations of burndown, velocity and control charts and cumulative flow diagrams, which have gained them many positive reviews.

**Codeclimate** is a platform that helps solve a problem that both our co-founders experienced as software engineering leaders: how can you ensure software quality even as you move further away from the code. Their ethos is about helping engineering teams to drive continuous improvement across people, process and code. Some of the visualizations offered by Codeclimate are Throughput, Cycle time, Rework and Velocity.

**Microsoft Workplace Analytics** does not measure directly software engineering, instead, its focus is on tracking overall engagement and measuring workers' productivity. They use data like the amount of time spent answering emails, the number of notifications received in a day and how many messages have been exchanged with colleagues. This data can then be used to enhance process efficiency and effectiveness and to create and validate effective workspace planning initiatives.



The average workweek span for employees was 43.8 hours.

On average, employees attended 14 meetings each week.

On average, employees sent 39 emails each week.

**Waydev** is a US-based Git Analytics Platform provider which offers similar measures as the past described. Waydev's focus is evenly distributed across the code-level metrics and the code review process. They discovered a way the code review process can be quantified into actionable metrics, which can enhance an engineering manager's visibility over his team collaborative activity. These insights can be used to spot a blocked pull request, incentivize engineers to collaborate more or adjust the workload. Waydev is designed to import information from popular Git applications like GitHub, GitLab, Bitbucket, and Azure DevOps. On the 3rd July, Waydev revealed that it had suffered from a data breach. In the last section of this report, I will address the issue of ethicality and threats that these measuring platforms can cause.[9]

## Which algorithms do these platforms use?

Behind the metrics and platforms introduced above, there are **algorithms**. In this context, I refer to an algorithm as a finite sequence of well-defined, computer-implementable instructions to perform a computation.

### CoCoMo (Constructive Cost Model)

A regression model based on the number of lines of code. It is a procedural cost estimate model for software projects and often used as a process of reliably predicting the various parameters associated with making a project such as size, effort, cost, time and quality.
According to CoCoMo, there are two main parameters which define the quality of any software outcome:

- Effort, which refers to the amount of labour that will be required to complete a task.
- Schedule, which means the amount of time required for the completion of the job.

The most complex and detailed implementation of CoCoMo also computes four cost drivers: product, hardware, personnel and project attributes. Each attribute is subdivided, each subdivision gets a rating, and those ratings are then combined together to compute the effort value. By computing these attributes in each step of the software development we CoCoMo achieves to obtain a precise value for effort.

### Halstead's Volume:

One of the three inputs of the maintainability index is Halstead's volume. I will now proceed to describe the algorithm required to calculate this Halstead metric.

The unit of measurement of volume is the common unit for size "bits". It is the actual size of a program if a uniform binary encoding for the vocabulary is used.

$$\text{Volume} = \text{Size} * (\log_2 \text{vocabulary}) = N * \log_2(n)$$

Where N = Number of operator occurrences + Number of operand occurrences
n = number of unique operators occurrences + number of unique operand occurrences

**WMFP (Weighted Micro Function Points):**

WMFP is a modern software sizing algorithm that is considered an alternative to measures like Cocomo and Halstead's complexity as it produces more accurate results while requiring less configuration and knowledge from the end-user. The WMFP algorithm uses a three-stage process:

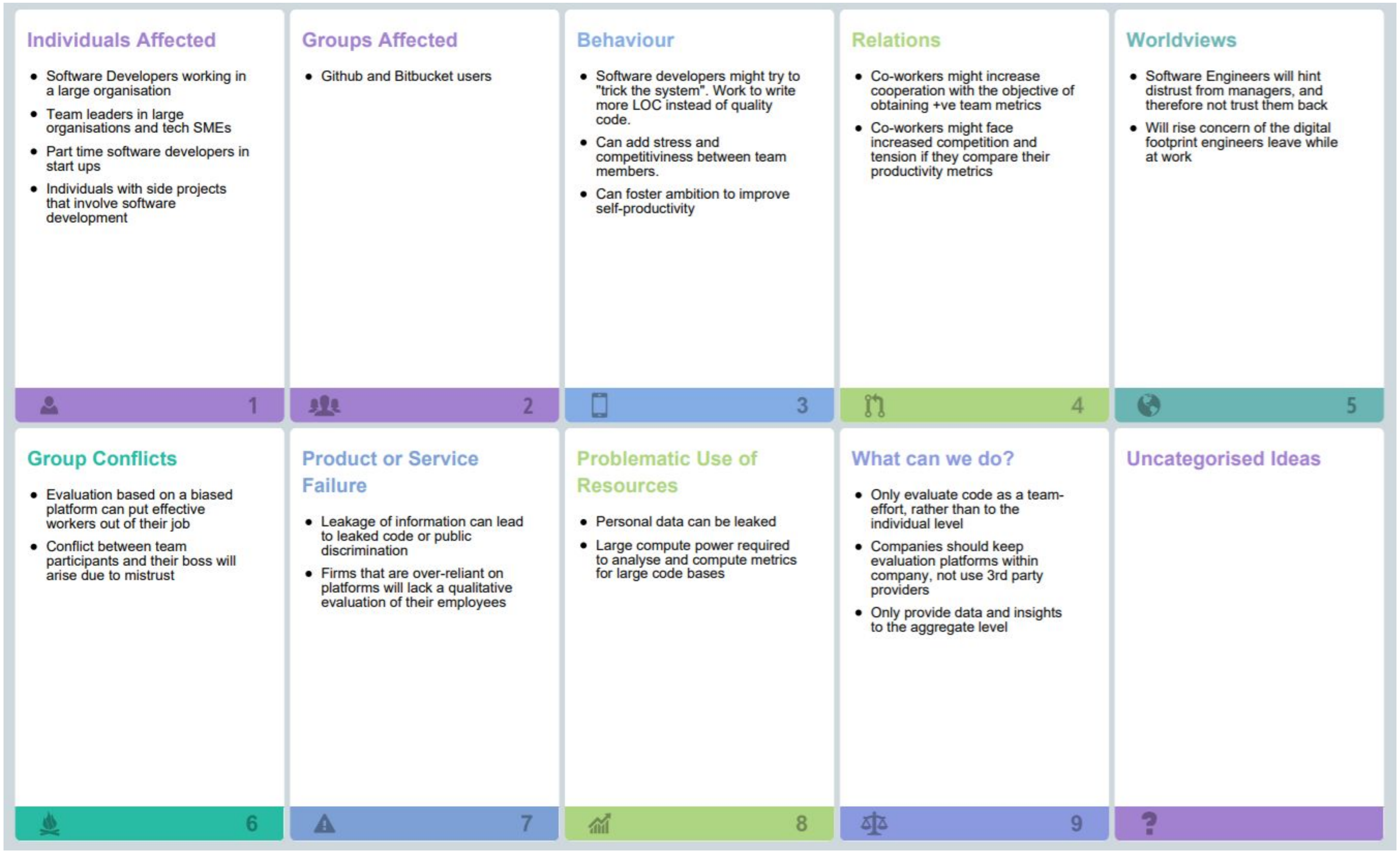
1) function analysis
2) APPW transform
3) result translation.

A dynamic algorithm balances and sums the measured elements and produces a total effort score.

## Is all this measuring ethical?

In order to address this question, I have used the Ethics Canvas, developed by a team of researchers in the ADAPT Centre. Summarizing, I believe that software engineering should be measured using aggregate data of all team members, and not to the individual level. I also consider that results displayed by platforms should not be compared between teams, as each different user story and project has a different complexity and type of requirements.

While discussing this project with an experienced software developer, he mentioned a personal story: "Yesterday I spent 5 hours fixing one line of code that was vital for the functionality of the whole project and none of my other co-workers was qualified to solve it. My work during those 5 hours was very appreciated by my technical lead, but any of your suggested algorithms would come to the conclusion that I have spent 5 very unproductive hours." This story illustrates one of the many situations that algorithms still are unable to capture, and therefore the importance of both quantitative and qualitative measuring techniques.



### Individuals Affected

- Software Developers working in a large organisation
- Team leaders in large organisations and tech SMEs
- Part time software developers in start ups
- Individuals with side projects that involve software development

1

### Groups Affected

- Github and Bitbucket users

2

### Behaviour

- Software developers might try to "trick the system". Work to write more LOC instead of quality code.
- Can add stress and competitiviness between team members.
- Can foster ambition to improve self-productivity

3

### Relations

- Co-workers might increase cooperation with the objective of obtaining +ve team metrics
- Co-workers might face increased competition and tension if they compare their productivity metrics

4

### Worldviews

- Software Engineers will hint distrust from managers, and therefore not trust them back
- Will rise concern of the digital footprint engineers leave while at work

5

### Group Conflicts

- Evaluation based on a biased platform can put effective workers out of their job
- Conflict between team participants and their boss will arise due to mistrust

6

### Product or Service Failure

- Leakage of information can lead to leaked code or public discrimination
- Firms that are over-reliant on platforms will lack a qualitative evaluation of their employees

7

### Problematic Use of Resources

- Personal data can be leaked
- Large compute power required to analyse and compute metrics for large code bases

8

### What can we do?

- Only evaluate code as a team-effort, rather than to the individual level
- Companies should keep evaluation platforms within company, not use 3rd party providers
- Only provide data and insights to the aggregate level

9

### Uncategorised Ideas

?

## What next?

The desire for companies to measure, quantify and improve the performance of their software engineering effort has lead to the creation of extensive research and a lucrative market of measuring platforms. This report covers a sample of the metrics, platforms and algorithms available to measure software engineering.

The objective of measuring software engineering is to maximise worker's or individual's productivity and efficiency. Any of the metrics and platforms now available in the market can be optimized for by the engineers, therefore it's important to measure many simultaneously and gather a combined score. In my opinion, qualitative metrics such as peer reviews have to be used in combination with metrics to obtain a real picture of the performance of software engineers.

With regard to the ethicality of measuring software engineering, it is important to note that employees in other industries are measured: teachers on the results achieved by their students, builders on the progress of a building, lawyers on the number of closed cases, salespeople on their sales numbers. In this context, why should software engineers not be measured?

## Bibliography

1 Bauer, Friedrich L. (1972). "Software Engineering". Information Processing. p. 71.

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

3 https://evansdata.com/press/viewRelease.php?pressID=278

4 https://www.extremeuncertainty.com/agile-metrics-ultimate-guide/

5 https://radon.readthedocs.io/en/latest/intro.html

6 https://www.atlassian.com/incident-management/kpis/common-metrics

7 https://homes.cs.washington.edu/~rjust/publ/google_coverage_fse_2019.pdf

8https://www.pluralsight.com/blog/teams/5-developer-metrics-every-software-manager-should-care-about

9 https://www.blueoptima.com/blog/the-financial-and-reputational-cost-of-a-data-breach