Measuring Software Engineering Report

Introduction

This report is about measuring software engineering. We are going to look at different ways we measure software engineering and we would get a conclusion for whether we should measure the performance of software engineers at the end of the report. The report would consist four sections:

1. How one can measure engineering activity.
2. The platforms on which one can gather and perform calculations over these data sets.
3. Various kinds of computation that could be done over software engineering data, in order to profile the performance of software engineers.
4. The ethics and legal or moral issues surrounding the processing of this kind of personal data.

Measuring engineering activity

Introduction

In this section, we are going to look at how to measure software engineering. But first, we need to understand: What is software engineering activity?

Software productivity can be defined as the ratio between the functional values of software produced to the efforts and expense required for development. [1] In the software engineering field, it specifically refers to the productivity of software engineers.

Ways to measure software engineering activity

There are many ways to measure productivity, however, most managers use two types of metrics:

- **Size-related metrics** indicating the size of outcomes from an activity. For instance, the lines of written source code.
- **Function-related metrics** represent the amount of useful functionality shipped during a set period of time. Function points and application points are the most commonly used metrics for waterfall software development, while story points are the usual metrics for agile projects.

Here are some metrics that many organizations use to measure software development productivity [2]:

- Lines of code per staff per month
- Function points per staff per month
- Story points per staff per month
- 360-degree peer evaluations
- Engineering leader evaluations
- Task-completion predictability
- Test cases passed
- Defect counts
- Cycle times

Abi Noda, Senior Product Manager at GitHub, often sees companies relying the following metrics, which he calls the “flawed five”:

1. Commits
2. Lines of code
3. Pull requests
4. Velocity points
5. “Impact”

Platforms

Introduction

In this section, we are going to look at platforms on which one can gather and perform calculations over these data sets. And we might be able to measure the productivity of a software engineer using these platforms.

There are different types of tools to measure employees' productivity, each has their pros and cons. The primary categories to consider include [3]:

- Project management tools
- Time tracking tools
- Team productivity & collaboration tools
- Results dashboard tools
- Specific reporting tools

There are some software applications that cross between these groupings, but in general, their strengths tend to lie in one area only.

Different types of tools

Project management tools

Project management tools have evolved greatly since the old waterfall, gantt charts that might immediately come to mind. These days, the best project management tools are based on Agile methodologies and also include some collaboration features. Their intention is to help employees manage their work and get things done.

Pros: These types of tools are great for businesses that work by projects, and already have defined processes for the way things should be done.

Cons: They can over-complicate simple tasks and are not as useful for less predictable types of work. They can also add to your team's workload as many of them require tasks to be entered with a lot of

detail and employees can spend more time fiddling with them than getting work done.

Time tracking tools
Since the rise of remote work and distributed teams, time tracking tools have risen in popularity. Essentially they either ask employees to report in on how they've spent / are spending their time, or they collect this information automatically by tracking activity on the employee's computer.

Pros: They are generally based on hard data and allow managers to keep a close eye on exactly what their team members are doing. Can be good for contractors or where management resources are stretched thin.

Cons: These types of tools are generally terrible for morale. They send a message to employees that they can't be trusted and are effectively just an hourly resource.

Team productivity & collaboration tools
Team productivity and team collaboration tools are ones which help team members plan their work and share their work with colleagues. (Chatting apps are also an important part of most teams toolkit, but that's not what we're talking about here.) Team productivity tools are generally not based on complex project structures, but instead are more concerned with the impact they have on team members. Some also include social features like rewards and recognition for getting things done.

Pros: These types of tools can unite teams that are not physically together and/or are not working on the same types of things. They use peer accountability to encourage high performance in teams, but they are also flexible enough to include all types of work. They help individuals get clarity about their work and focus on getting the important things done.

Cons: These tools are not based on hard data, but instead rely on employee inputs.

Results dashboard tools

Dashboards give real-time visibility to the key metrics for the business. They are generally third-party applications which integrate with the key software used by the business (e.g. the accounting system, sales system, web analytics, etc.)

Pros: Knowing the real numbers of a business or team can help managers to spot trends and make better decisions. These kinds of tools are great when the team is relatively small and each team member knows exactly how they contribute towards the results.

Cons: Once teams become bigger, employees can disengage from aggregated metrics and feel they have little relevance. Dashboards are usually also based only about outputs - which are sometimes quite disconnected from the employee's actual inputs. For example, a development team could release fantastic new features which are entirely bug-free, but yet that is unlikely to show up on a dashboard as new sales or better user retention for many months.

Specific reporting tools

Depending on the type of work your teams perform, there are all kinds of industry and function specific tools that can help track and report on outcomes. This could be reports generated from your phone systems to show calls made and call times for each operator, it could be sales activity reporting, support ticket reporting, development reporting, or even employee sentiment reporting.

Pros: These types of tools can provide specific and granular data that can help managers develop team members. They are based on real data and often include leading indicators (which can be more useful than lagging indicators).

Cons: As these tools usually have a narrow and data-driven focus, they can miss some of the more qualitative elements of productivity. For example, one employee could be closing more support tickets than anyone else but might be destroying goodwill. A developer could be pushing lots of commits but that doesn't say anything about the quality of their work.

Data Computation

Introduction

This section is about various kinds of computation that could be done over software engineering data, in order to profile the performance of software engineers. In my report we would focus on two techniques:

1. Counting lines of codes
2. Software engineering algorithms measuring concepts

Different techniques

Counting lines of codes

It's easy to understand, literally, counting lines of codes that a software engineer has written to determine their productivity. This is an old and straightforward technique, but infamous at the same time. It has had a long journey throughout software development history and its advantages—when used in the right context—are beginning to be better understood. Measuring the Lines of Code per day metric can give you a high level view of the size of a software system, how the code base is changing and potentially the complexity of that code base. With that said, the metric can yield incorrect behaviors, expectations, and results, when used in the wrong context [4].

Pros:

- **It measures code length**, which is a viable measure of ballast - if we were managing a team of programmers, we would very much want to keep track of the ballast it creates or removes.
- **It tells you how large your system is** - which is great for judging complexity and resources and helps you prepare the next developer for working on a code base.
- **It can help you predict defect density** - LoC can be useful as a value to derive other metrics, such as predicting defect density in your software.

- **It's suitable for automation of counting** - since Lines of Code is a physical entity, the manual counting effort can be easily eliminated by automating the counting process.
- **It's an intuitive metric** - LoC is great for measuring the size of software because it can be seen and the effect of it can be visualized, even by non-technical parties.
- **It's the most used metric in cost estimation** - studies do show a rough correlation between LOC and the overall cost and length of development. The lower your LOC measurement is, the better off you probably are.
- **It's inexpensive** - LoC is one of the simplest software metrics and is cheap to collect.

Cons:

- **It's viewed as a vanity metric** - extra lines of code are not a sign of progress. Vanity metrics are antithetical to proper software engineering, which is about reducing complexity and reducing lines of code.
- **It's an easy metric to game** - the number of Lines of Code per day doesn't tell you the actual value delivered. Just like the number of commits and pull request count, it's very easy to game. Just create more lines of code.
- **It encourages negative behaviors** - even if it's not gamed, a rise in LoC doesn't indicate more productivity, output or value delivered. Instead, this causes bloat in the code review process and creates unnecessary overhead across the team.
- **It promotes less efficient code** - some think it isn't useful to measure the productivity of a project using only results from the coding phase, which usually accounts for only 30% to 35% of the overall effort.
- **It increases code complexity** - any good software developer knows that good code or refactors are actually trying to produce less code complexity, not more.
- **It doesn't balance cohesion with functionality** - experiments have shown that effort is highly correlated with LoC, but functionality is less so. That means that skilled developers may be able to develop the same functionality with far less code, so one program with less LoC may exhibit more functionality than another similar program.
- **It's a poor productivity measure of individuals** - a developer who develops only a few lines may still be more productive than a developer creating more lines of code - even more: some good

refactoring like “extract method” to get rid of redundant code and keep it clean will mostly reduce the lines of code.

- **It doesn’t take developer experience into account** - the implementation of a specific logic can be different based on the level of experience of the developer. An experienced developer may implement certain functionality in fewer lines of code than another developer of relatively less experience does.
- **It doesn’t take into account the difference in languages** - since it only measures the volume of code, you can only use LoC to compare or estimate projects that use the same language. The same code could be written a dozen different ways, each performing the same action, and each taking up different numbers of lines of code.
- **It doesn’t take into account code that is automatically generated by a GUI tool** - with the advent of GUI-based programming languages and tools such as Visual Basic, programmers can write relatively little code and achieve high levels of functionality.
- **There is a lack of LoC standards** - there is no standard definition of what a line of code is and questions continue to arise: Do comments count? Are data declarations included? What happens if a statement extends over several lines?
- **It affects the programmer psychology** - a programmer whose productivity is being measured in lines of code will have an incentive to expand their code with unneeded complexity.

Software engineering algorithms measuring concepts

When it comes to the measuring of algorithms, especially resources usage, the two most common measures are [5]:

- Time: how long does the algorithm take to complete?
- Space: how much working memory (typically RAM) is needed by the algorithm? This has two aspects: the amount of memory needed by the code (auxiliary space usage), and the amount of memory needed for the data on which the code operates (intrinsic space usage).

We have learned time complexity and space complexity in lectures, they can represent the efficiency of the algorithm in a convenient way.

Pros: The efficiency of algorithms can be easily summarized and compared.

Cons: I haven't really found any information related to the disadvantages of complexity analysis on the internet, and I can't think of any big disadvantages that complexity analysis have.

Ethics and moral issues

Introduction

This section is about the ethics and legal or moral issues surrounding the processing of personal data related to the performance of software engineers as they go about their work.

Should we measure?

So, should software engineers' productivity be measured? Different people have different opinions.

In a paper called "Summarizing and Measuring Development Activity", it suggested that [6]:

"As many of our participants indicated, development activity is impossible to measure, and it might even be dangerous to measure it since measures could lead developers to game a system rather than work towards the "goodness of the code base". One of the challenges is the prevalence of invisible work and articulation work in software development that cannot be measured in terms of lines of code, number of commits, or number of issues."

My opinion

In my personal view, it is alright to use different methods to measure different metrics to get more data to know more about how the software engineers are working and to find out if they need more help. But I don't think it is a good idea to tie the performances of the software engineers to the measurement directly. It would incentivize software engineers to put their focus on "how to game the system" rather than "how to be more efficient and really solve the problems". Sometimes it is also hard to tie some metrics to a specific individual as they might be contributed by several team members. Last but not the least, software engineers' privacy need to be respected, so if there are measurements

towards their productivity then it should made sure that these measurements haven’t crossed the line.

Reference

[1] https://www.infopulse.com/blog/top-10-software-development-metrics-to-measure-productivity/
[2] https://www.getclockwise.com/blog/measure-productivity-development
[3] https://www.saasgenius.com/blog/how-find-best-software-measuring-employee-productivity
[4] https://waydev.co/lines-of-code-per-day/
[5] https://en.wikipedia.org/wiki/Algorithmic_efficiency
[6] Christoph Treude, Fernando Figueira Filho, Uirá Kulesza. Summarizing and Measuring Development Activity.