Measuring Software Engineering

John Burke
18326420

Contents

Introduction

Software Engineering is a rapidly changing field. New methodologies and dogmas are proposed every year, offering to increase productivity and make developing code easier. While it is accepted that Agile development and DevOps have drastically shortened the systems development life cycle, how do we know this?

The simple answer is that we measure the productivity of developers. But this itself raises more questions such as “How do we define productivity?” and “What exactly do we measure?”. This report hopes to provide a brief summary of proposed methods and technologies, along with the advantages and disadvantages of each. Algorithms can also be used, as we will see, to help aid us on our discovery of the most imports metrics of software engineering. Lastly, we will discuss the ethics of measuring software engineering, examining the line between enhanced productivity and personal privacy violation.

While each method mentioned here has its advantages and disadvantages, it should be stressed that there is no ‘one size fits all’ method, as different situations call for different methods. While this data collection and analysis can lead to an increase in worker productivity, it is a foolish mistake to believe that it can solve all the problems of developing software. As Fred Brooks argues, there is no “**Silver Bullet”**, that “there is no single development, in either technology or management technique, which by itself promises even one order of magnitude improvement within a decade in productivity, in reliability, in simplicity[1]”. So, while we may be able to monitor the effectiveness of developers, no solution can be offered to drastically solve or fix the problems inherent to software engineering.

[1] http://worrydream.com/refs/Brooks-NoSilverBullet.pdf

Types of Measurement

Given that every part of the development cycle is recorded, there is an abundance of data available. This process can be automated with tools and used to create models attempting to measure the productivity of a project, individual or team. The measurement of the software process is a crucial component in the endeavour to reach highly capable and productive projects[2]. As such this data is essential to the improvement of the software process. There are many ways in which the process can be measured, some of which will be explored in this passage.

Source Lines of Code

Lines of Code was one of the first widely used metrics for the size of software systems and the productivity of programmers. It is a metric used to measure the size of a software program by counting the number of lines in the programs source code. This can be used to estimate the amount of effort required to write a program as well as to estimate a programmer's productivity.

This can be split into two metrics: Physical lines of code (LOC); and Logical lines of code (LLOC). The former of which includes comments and whitespace and the latter counting statements.

The main advantage of this metric is how easy it is to collect data for. This data can then be visualised for further analysis.

This metric isn't as popular as it was, due to its glaring disadvantages. This metric only measures the coding phrase of software development, ignoring the test and planning phases. Similarly, it is a poor judgement of individuals, as one programmers' style might be more compact than another's. A skilled developer may be able to develop the same functionality with far less code.

[2] http://www.robertfeldt.net/publications/unterkalmsteiner_2011_tse_spi_sysrev.pdf

Commit Frequency

Measuring commit frequency is very similar to measuring lines of code, where we measure the average number of times each developer integrates tested, working code into the master branch.

The aim of this metric is to encourage good habits: To increase code integration frequency; to encourage small changes over big ones; and, if used right, to support greater code transparency and collaboration through more frequent code reviews.

Again, one of the main advantages of measuring commit frequency is its ease. Most version control services track the number of commits made by individuals, which can be accessed visually or through APIs.

However, it should be noted that counting the number of commits doesn't tell you the value delivered. Like measuring lines of code, it is very easy to game. Even then, a rise in commits doesn't indicate more productivity, output or value delivered.

Work Hours

Another simple metric to measure is the hours put in by a developer. Easy to measure, the main idea here is that more hours worked lead to more work done, that if you worked 10 hours instead of 8 you would get 25% more work done.

Studies show however that after a point, the inverse occurs. Working more than 40 hours a week leads to a drop in productivity[3]. As workers become over worked, productivity drops, leading to sloppier code and a decrease in morale as workers become stressed and burnt out.

Cycle Time

This is the first of the Agile metrics we can measure. Cycle time, simply put, is the total time that elapses from the moment when a piece of work is started on a task until its completion.

---

[3] https://cs.stanford.edu/people/eroberts/cs201/projects/crunchmode/econ-hours-productivity.html

This does not begin from the creation of the task, only from when work is started on it. The faster the cycle time, the quicker new features can be introduced for the end users.

Tracking the average time taken per cycle allows teams to estimate the cycle time of future tasks, improving the transparency between the team and business owners. This allows teams to manage expectations and avoid obvious surprises down the line.

This is simple to track using online task boards, or even physical boards if the dates are recorded. In a Kanban board, the time measured is from when the task is moved into the "Doing" column until it is moved into the "Done" column.

This data can be processed into rolling averages, to visualise the productivity of teams over a long period of time.

Team Velocity

The simplest definition of a team's velocity is the measure of how much functionality a team delivers in a sprint. To measure the velocity, simply add up the estimates of user stories, features, backlog items and requirements successfully delivered in an iteration[4]. As Agile delivery cycles are very small and short, a team's velocity can be tracked quite frequently.

This data can be visualised simply with a bar chart: Showing the estimated total points of the Sprint per iteration. Typically, this will stabilise with time and can be used to help plan future iterations, again managing expectations of product and business owners.

Tools

Now that we know what data we are looking for; how do we get it? And once we do, what do we do with it? There are many tools out there that track developer's and team's work, most of which can be accessed through API's or provide their own data analysis which can be

[4] https://digital.ai/resources/agile-101/agile-scrum-velocity

viewed. Other times, the data obtained can be fed into other programs, which can provide insights and visuals of the data.

GitHub and Git

Git is an open source distributed version control system[5]. GitHub provides hosting for this version control as well as providing other services such as bug tracking, feature requests, task management and continuous integration. It is one of the most popular git hosting platforms with over 40 million users.

It's users provide a wealth of information which can be accessed using the GitHub API[6]. Besides allowing programs to create their own git requests and other actions using the API, it also allows programs to gather public data about users, teams and projects.

Data can be obtained from the GitHub API endpoint: https://api.github.com

This includes:

- **Project Commits** - Data about a project's commits can be fetched.
    - GET /repos/:owner/:repo/git/commits/:commit_sha
- **User Commits/Pull Requests** – Data about a user's events can be fetched.
    - GET /users/:username/events
- **Release Data** – Data about a project's releases which could be used to work out the cycle time.
    - GET /repos/:owner/:repo/releases
- **Project Cards** – Data about a project's tasks including the task's state.
    - GET /projects/columns/:column_id/cards

Some of this data can also be viewed graphically through GitHub's project insights tab. This shows a user's commit frequency, represented as the number of commits per calendar day.

---

[5] https://git-scm.com/

[6] https://developer.github.com/v3/

This insights tab visualises the number of contributors to a project, the traffic, forks, branches and SLOC.

This data can be used to gather insights to an individual developer or team's work, covering most of the metrics mentioned above. As stated above, GitHub provides its own insights to most of the data, however data obtained through the API can be processed using other tools and software.

JIRA

JIRA is a propriety issue tracking product developed by Atlassian that allows bug tracking and agile project management[7]. JIRA's project management component allows users to track releases, workflows and issues.

JIRA's API is built into your product, so the endpoints are on your own private JIRA server. The API provides support for data collection of agile metrics

- **View tasks and issues assigned to an individual** – Can view all the issues assigned and completed by an individual, useful for finding a developers velocity.
    - GET /rest/api/2/search/?jql=assignee=:name
- **View tasks finished in a timespan** – Can view all tasks finished in a timespan which can be used to work out team velocity.
    - GET / rest/api/2/issue/search?jql=duedate=:"xxxx-xx-xx"

This data, like GitHub's API, can be used to measure some of the agile metrics mentioned in the above section. This software, however, is not open source and costs money and time to integrate.

[7] https://www.atlassian.com/software/jira

Personal Software Process (PSP)

The Personal Software Process (PSP) provides engineers with a disciplined personal framework for performing software work[8]. It provides a toolkit of methods, forms and scripts that help and show developers how to plan, manage and measure their work.

Developed by Watts Humphrey while working at Carnegie Mellon University, the PSP is intended for developers who wish to improve their personal software development process.

PSP is split into four levels:

1. **PSP 0 –** Personal measurement, basic size measures and coding standards.
2. **PSP 1 –** Planning of time and scheduling.
3. **PSP 2 –** Personal quality management, design and code reviews.
4. **PSP 3 –** Personal process evolution.

As forms and scripts are an integral part to this framework, there are a variety of tools available. **The Software Process Dashboard[9]** is an opensource initiative to create a PSP support tool. Originally developed by the United States Air Force, it has now evolved under the open source model. The dashboard supports:

- **Data Collection –** Time, defects, size and plan vs actual data.
- **Planning –** Integrated scripts, templates, forms, summaries and earned value.
- **Tracking –** Earned value support.
- **Data Analysis –** Charts and reports aiding the analysis of historical data trends.
- **Data Export –** To allow the use of other tools.

The open source model allows the dashboard to be modified and distributed at no cost.

The data collected and analysed by the PSP and supporting tools allows developers to evaluate their work and can suggest improvements.

---

[8] https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=5283

[9] https://www.processdash.com/

Algorithms

Algorithms are an essential component to both data production and analysis. While it's obvious that algorithms are needed to interpret trends in data, they can also be used to improve the quality of data collected; picking out only the most important metrics to record. Using such algorithms to analyse existing high-quality software projects can yield insights into which metrics we really should be monitoring.

Genetic Algorithms

A genetic algorithm is a method for optimising a given situation based on a natural selection process that mimics biological evolution.

This algorithm can be applied to many of the metrics mentioned in the previous section to find out the most prevalent metrics seen in high quality software projects. One study, using metrics such as SLOC, number of comments, number of methods amongst others, has found that after 20 generations, the main metrics correlating to a quality software project were: Number of comments and comment length (making development easier for other developers); white space and method name length (indicating the readability of the code); and the apparent complexity of methods[10].

Affinity Propagation

Affinity propagation is a clustering algorithm (groups similar entities together). The algorithm works by passing 'messages' between data points. Each data points sends out it's relative attractiveness to each other data point, which allows the receiver to indicate its ability to associate with the sender, creating a cluster of points.

This to can be applied to the software metrics discussed in order to indicate some of the most important metrics prevalent in high quality and successful software projects. This approach can be combined with a genetic algorithm to create another strategy to analyse software

[10] https://ieeexplore.ieee.org/document/1226139

metrics for predicting software quality. One such study, using the same metrics as described above, verified the correctness of one such algorithm, proving it can be used to obtain better metrics automatically for software quality prediction[11].

Ethics and Legality

In theory, employee data collection and analysis have no downsides: You can monitor your employees and get them to perform at their best. However, how much is too much? We must find the line between monitoring and the breaching of privacy.

This potential to significantly enhance the developer's productivity, while alluring, comes with the potential for misuse. Ethical oversight and constraints are required to ensure that an appropriate balance is reached. Since there is no absolute authority one can look to for guidance, it is up to both the data creators and the users to engage in these ethical considerations.

The users; project managers; data analysts and managers; must be aware of and comply with any applying laws. The European Union proposed and passed the General Data Protection Regulation (GDPR) in 2016, coming into effect in 2018. With this regulation came provisions and requirements relating to the processing of personal data of individuals located in the EEA[12]. This includes the right of an individual to view the data relating to them and to request the removal and deletion of such data. Penalties for breaking this include fines of up to €20 million, or 4% annual global turnover, whichever is higher.

Similarly, the users must keep into consideration the wellbeing of their workers and those they are monitoring. Constant and omnipresent monitoring can lead to a lack of trust among

[11] https://sss.bnu.edu.cn/~pguo/pdf/2008/y.pdf

[12] https://gdpr-info.eu/

workers and managements, to an increase in stress among workers and even decrease individual creativity[13].

However, the onus cannot be left entirely on the users. The data creators; the workers; and developers must play an active role in protecting their digital rights and privacy. Workers who are aware of their right to digital privacy and the role of the GDPR (in Europe) are more likely to report abuse, be it the monitoring of unnecessary and pervasive data, or a lack of transparency.

The ethics of big data, in the field of software engineering is a complex topic. While the monitoring and analysis of certain metrics can lead to an increase in worker productivity, we must be careful and transparent in the process, to ensure that the rights of the worker are kept intact.

[13]https://www.researchgate.net/publication/10702133_When_the_Presence_of_Creative_Coworkers_is_Related_to_Creativity_Role_of_Supervisor_Close_Monitoring_Developmental_Feedback_and_Creative_Personality

Conclusion

Software engineering can be measured in a multitude of ways, some better than others. However as stated in the introduction, while the measurement and analysis of such metrics can lead to an increase in performance of a team or developer, no great leaps of improvement can or should be expected.

The metrics mentioned can be monitored using a variety of tools. Many development tools used by developers record these metrics., which can be accessed through API's. These can be both open source and proprietary. The analysis of this data can be analysed by the same tools creating it, allowing teams and developers to interpret trends and improve their workflow.

Not all metrics are equal, as discussed throughout the paper. Algorithms can be used to pick out the most important and leading metrics of a software project. This itself can allow for a better software engineering measurement process.

Lastly, we discuss the ethics of employee data monitoring. While it is clear than monitoring can lead to an increase in performance, managers need to be aware of it's limits, be it through employee exhaustion and dissatisfaction, or breaking laws such as the GDPR. Employees themselves need to be aware of their rights and should play a major role in protecting their right to privacy, reporting any such violations.