MEASURING SOFTWARE ENGINEERING

Barry O Sullivan

18325155

Table of Contents

Introduction

The world that we live in today is rapidly evolving every day and the software industry is at the forefront of a lot of these changes. Software engineering production has been streamlined due to methodologies such as Agile development and DevOps. The speed at which the industry is developing comes with a large responsibility for software companies to measure their employees from a capability standpoint as well as assessing their own ethical values regularly.

This report will focus on explaining the various ways in which Software Engineering process can be measured. It outlines what platforms can be used to gather and process data as well as explain the variety of algorithmic approaches and the ethical concerns surrounding the software engineering process today.

How can Software Engineering be measured?

There are endless methods of gathering metrics from a software development team ranging from counting the lines of code written by a given developer on a given day to measuring how much of an impact each minute spent working by that developer has had towards a given project.

When approaching a basic metric such as how many lines of code are written by a given developer each day there are quite a number of factors that must be taken into account. Just because a developer has produced over 10,000 lines of code this week does not mean that these lines are optimised, bug-free, the best approach and easy to read. This output must be measured in parallel with the testing done on the code such as through methods of unit testing and code coverage to ensure all possible scenarios have been accounted for and handled correctly. Unit testing and Test Driven Development (TDD) can add an additional 15-35% increase in initial development time which may come across as substantial, but this increase in development time also brings with it a 40- 90% decrease in pre-release defects. From this same survey 78% of developers thought that TDD improves the overall productivity of the programmer and 92% believed that TDD yields higher quality code. (George & Williams).

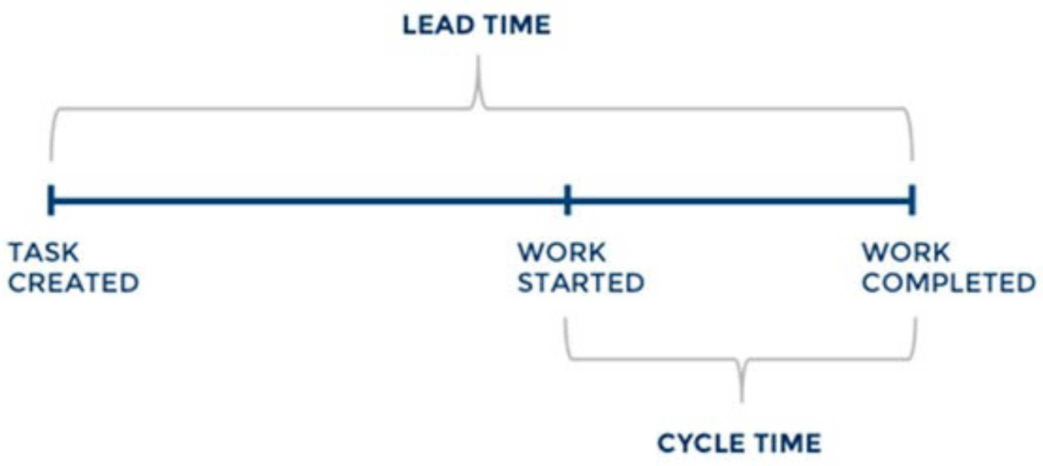
Agile Production Metrics

The actual writing of lines of code is not the only metric that can be empirically measured. Modern teams in the software engineering industry will use agile development processes and these metrics will provide them with valuable information they require for project planning. Examples of Agile production metrics are:

- Lead Time
- Cycle Time
- Active days
- Code Churn
- MBTF and MTTR
- Application Crash Rate (ACR)

Lead Time

This is a metric that is used to measure the time taken for ideas to go from inception to development. A lower lead time is optimal as it means that a team is very responsive to client requirements and changes.



Cycle time

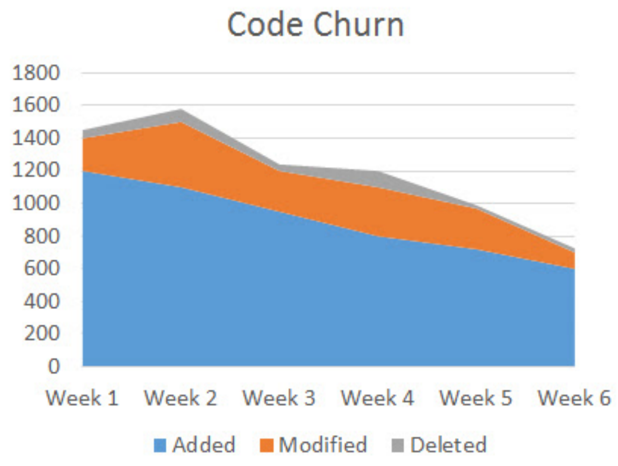
Cycle time is a measure of the elapsed time when work starts on an item until it is ready for delivery. Cycle time tells how long it takes to complete a task. It is generally grouped together with lead time. It is a helpful metric for informing clients when features will be completed, and it helps manage expectations effectively.

Active Days

Active days is the amount of time a developer spends writing code for the team. It is an important measure because it allows team members to see how long certain tasks take. Active days doesn't include any administration time it is strictly to do with how much time an employee is spending on the isolated task at hand. This is helpful because it allows managers to see how much time is being wasted during the development process and allows for improvement in future projects

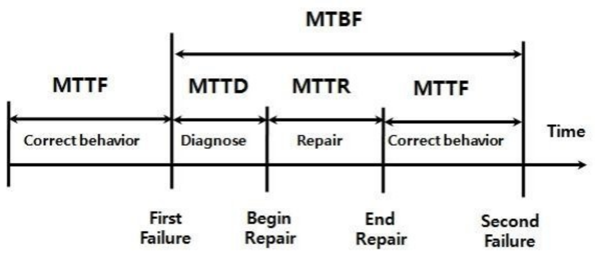
Code Churn

This is a measure of the number of lines of code that were modified, added or deleted from the code base over a specified period. If the code churn on a task increases dramatically it may indicate that the task requires more attention. The level of code churn will fluctuate over the lifetime of a task and ideally it will be steadily decreasing as the task nears completion.



MTBF and MTTR

Mean time between failures and mean time to recover are both used to measure a piece of software's performance in a production environment. Software failures are inevitable, but these metrics are used to quantify how well the software is able to recover from specific bugs.

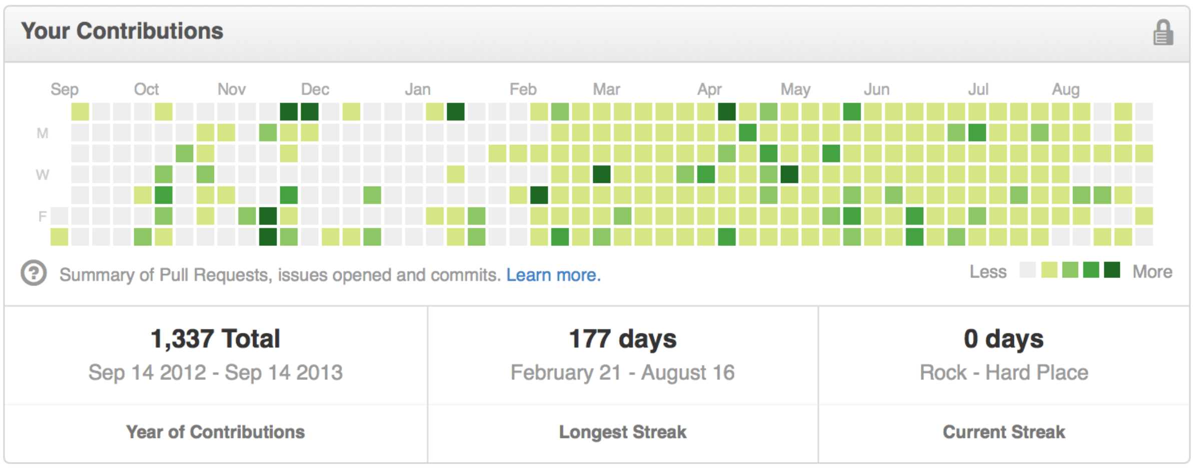


Application Crash Rate

This is a simple measurement used to keep track of how often the software crashes. It is calculated by dividing the application fails by the number of times it was used.

What platforms can be used to gather and process data?

A lot of companies and organizations today rely on GitHub to host their projects. GitHub, which has over 2.1 million repositories, is a useful resource for getting information about a project's development cycle and the people working on it. Statistics that can be gathered from a GitHub commit include the author of a commit; the date of a commit; the file or directory a commit was made in; the number of lines modified in the change; and what platform wrote the code (Linux, OS X, or Windows).



Summary of Pull Requests, issues opened and commits. Learn more.

1,337 Total	177 days	0 days
Sep 14 2012 - Sep 14 2013	February 21 - August 16	Rock - Hard Place
Year of Contributions	Longest Streak	Current Streak

If you would like to get an idea of the output from a given developer, look at their GitHub commits. These statistics can help you see how much of an impact someone has had on a project. The more code a developer has reviewed or worked on, the more important they might be. If there's a bug, it's likely that your most experienced developers will know how to fix it.

Automated tools that plug directly into a developer's IDE can be quite useful for analysing metrics within an organization and for the purpose of quality assurance. One such product is TestRail, which integrates directly with GitHub for code repository management, and Jira for project management. TestRail does not gather source code productivity metrics such as individual productivity by lines of code produced. Instead, it serves the purpose of allowing teams to monitor their testing metrics, providing user-friendly interfaces which allow both developers and project managers to closely monitor testing quality, test results, and other useful metrics relating to individual tasks they work on.

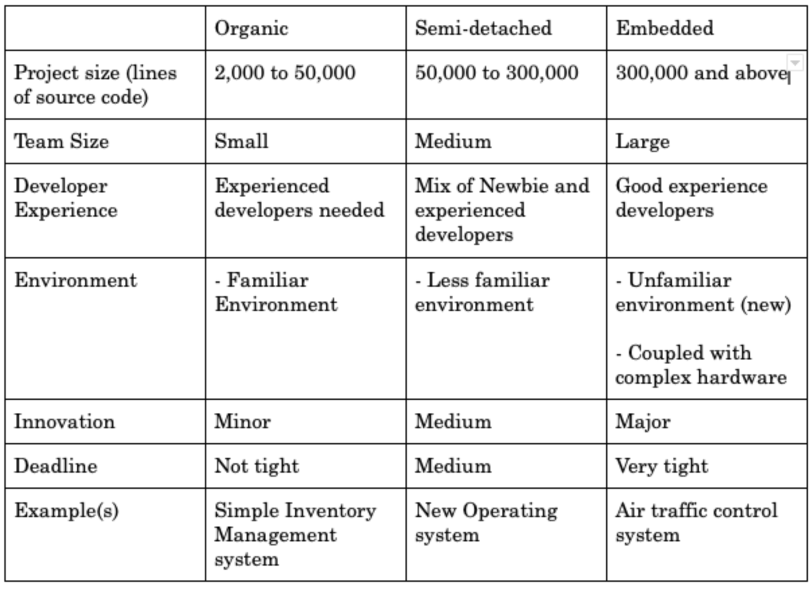
Code coverage can be an extremely useful tool for ensuring that your project has been thoroughly tested and thoroughly written. It's also a useful metric for ensuring that your developers are testing their code effectively, and aren't just writing (and rewriting) code without actually testing it.

You can use a tool such as Jira, which was developed by Atlassian, to track tasks created in sprints. The tool's UI has a dashboard and kanban layout. Tasks are displayed at the top of the board and board columns represent statuses: To Do, In Progress, and Done. Each task has a description box with instructions, an assignee field, and a backlog field. For example, if you want to create a new task in Jira, you first select the type of task or an issue that you want to resolve. You then enter data such as the issue summary (what the task is), severity (how important it is), priority (how quickly it needs to be resolved), status (whether it's accepted), due date (when it should be completed), assignee (who should work on it), and comments (what else you want to tell about it). Atlassian Jira provides charts and metrics to help product managers manage requirements. One of these charts is the burndown chart, which shows the total work remaining and a team's likelihood of achieving its sprint goal. Jira also integrates with other Atlassian products such as GitHub, allowing for developers to get their code reviewed really quickly and reduce overhead time spent on getting code from development to production.

What Algorithms can we use?

COCOMO Model

COCOMO (Constructive Cost Model) is a regression model which assists in determining the cost, effort, and development time required to produce a software project. This model was produced by Barry W. Boehm. Software projects which use the COCOMO model are classified into three categories which are organic, semi-detached and embedded. The definition of each is found below:



	Organic	Semi-detached	Embedded
Project size (lines of source code)	2,000 to 50,000	50,000 to 300,000	300,000 and above
Team Size	Small	Medium	Large
Developer Experience	Experienced developers needed	Mix of Newbie and experienced developers	Good experience developers
Environment	- Familiar Environment	- Less familiar environment	- Unfamiliar environment (new) - Coupled with complex hardware
Innovation	Minor	Medium	Major
Deadline	Not tight	Medium	Very tight
Example(s)	Simple Inventory Management system	New Operating system	Air traffic control system

The different categories influence the algorithms used during the calculation of the estimates. The COCOMO model has three different models which allow the software manager to specify the granularity of the estimates. The three types are: Basic, Intermediate and Detailed.

- Basic Model- This model is quite limited and restricted—it uses kilo lines of code for calculations which are too simplistic. The model does not take into account the other factors for the estimation.

- Intermediate Model- This model is an improvement of the basic COCOMO model and takes into account cost drivers which affect effort, time, and cost during the development phase of a project. The cost driver is a term used to define factors which affect these three aspects.

- Yet again, this approach builds on the basic and intermediate models—and incorporates the Waterfall model, at least in terms of cost estimates. This is the most detailed model available.

Machine Learning

"Machine learning is a branch of artificial intelligence and computer science which focuses on the use of data and algorithms to imitate the way that humans learn, gradually improving its accuracy." (IBM,2022)
Nowadays machine learning is used extensively. It is used to power things from self-driving cars to spam filtering. The applications of machine learning are endless. In order to use machine learning, one must have a large data set to work with. One of the side effects of using a version control system is that it produces a large data set regarding software engineering. One application of machine learning is to process the data from version control systems to measure software engineering productivity. A highly productive software engineer is one that produces more source code and of better quality. With the help of Artificial Intelligence, complex code structures go through complex algorithms resulting in a quantification of the quality and quantity of source code produced.

Machine learning is leading the way in Software Development evolution. This is shown clearly in this quote by Google chairman, Eric Schmidt, "Google's self-driving cars and robots get a lot of press, but the company's real future is in machine learning, the technology that enables computers to get smarter and more personal."

Ethics

Whilst some of the methods discussed above can be extremely useful for an organisation with a software development branch they also bring with them some serious ethical concerns and implications.

Introducing systems or methodologies that cause developers to be under constant watch and measure from the second they walk into the office does not create a comfortable environment in which to work and also strongly hinders productivity. Instead of focusing on the task at hand, many developers may feel they are under direct pressure to meet timing and productivity level quotas. Developers may channel all of their resources into meeting these quotas and produce a poor end product that does not meet the desired standards of the project managers and customers. For this reason the project managers within an organisation must think carefully about how they want to roll out and make use of their desired measuring metrics.

Having developers under constant monitor within a workplace also introduces some ethical concerns such as a quite serious invasion of privacy. Whilst it may be useful to have an automatic quantitative software development measuring tool built in to developers IDE's to track coding metrics, organisations must be careful not to take this too far and start infringing on people's privacy. Proceeding further than this and attempting to gather metrics from a developers daily life within the workplace such as how long their breaks take, how frequently they use the bathroom, who they talk to etc. may introduce an extremely hostile and somewhat dictatorship-like environment that will serve a negative impact on the productivity of developers rather than increasing their productivity.

It is for reasons like this that software giants such as Google, Facebook and Amazon have transitioned their workplaces into enjoyable and comfortable places to work rather than a strictly enforced organisation that is solely focused on producing products at the highest rate. Giving developers a comfortable platform upon which to work will no doubt increase their productivity significantly more so than the latter option.

Conclusion

For the reasons discussed above, I believe that determining software developer productivity is more complicated than it appears on the surface. The field of software engineering is constantly changing, and as a result, definitions of productivity that may have held true in the past are becoming somewhat obsolete in modern software engineering. methodologies such as Agile Production Metrics have helped project managers gather metrics that they can use to measure their teams' productivity. This approach has also encouraged a fast-paced environment for developers to work in, which makes it easier to stay productive and reduce the need for scrutinizing metrics—although there is plenty of time to do so when necessary.

Bibliography

George, B., & Williams, L. (n.d.). An Initial Investigation of Test Driven Development in Industry.

Education, I., 2022. *What is Machine Learning?*. [online] Ibm.com. Available at: <https://www.ibm.com/cloud/learn/machine-learning> [Accessed 2 January 2022].