**Measuring Software Engineering**

**Methods of Measuring Software Engineering Activity**

Measuring the productivity of software engineers is a topic of prime interest within the industry. Companies endeavour to do this for several reasons including to develop competitive analyses and benchmarks, to track progress over time, to reward high performers, to determine resource allocation and to identify and spread more productive development processes across the organisation. There are various metrics used and I will go into depth about each of the most common ones now.

Perhaps the most common and basic metric used is the number of lines of code written by a software engineer in a specific time frame (week, month etc). This metric is not very good for measuring productivity, especially when used as a single metric. First, it’s an easy metric to game. If engineers are being assessed on the number of lines of code they are writing, naturally they will write as many lines of code as possible. This is antithetical to proper software engineering, which is about reducing complexity and reducing lines of code. It’s also just a poor productivity measure for individuals, a more skilled engineer may write less code for a specific problem than a less skilled engineer would for that same problem. Similar metrics include counting the number of commits and pull requests a software engineer makes, which are also relatively easy to game by splitting work that would be used for one commit/pull request into several commits/pull requests. This also acts as a disincentive for software engineers to tackle large, hairy problems.

A more meaningful metric that companies use is Impact. Impact is a way to measure the amplitude of code changes that are happening in a more complex manner than measuring raw lines of code. Impact attempts to answer the question: "Roughly how much cognitive load did the engineer carry when implementing these changes?". Impact usually takes the following into

account: what percentage of edits is work to old code, the number of edit locations in the code, the number of files affected, the severity of changes

when old code is modified and how this change compares to others from the project history. Let’s consider a hypothetical situation. One software engineer touches three files, at four different edit locations, where they add twenty lines of code while deleting 30 lines of code. Another developer produces one hundred lines of code in a single file. The significance of each contribution can't be boiled down to just the amount of code being checked in. Even without knowing specifics, it's likely that the first set of changes were more difficult to implement, given that they involved several spot-edits to old code. Even without knowing the gravity of changes or comparing to previous changes, it's safe to say that the first contribution was more 'impactful,' and therefore carries a higher impact.

Companies also use metrics to measure how productive teams of software engineers are when working together. Sprint burndown is one of the key metrics used for agile scrum. A burndown report communicates the complexion of work throughout the sprint based on story points (estimate of the overall effort required to fully implement a piece of work). By tracking this metric, you can obtain important insights. Consistent early sprint finishes can signify lack of scheduled work for one sprint, while consistently missed sprint deadlines can indicate a gap in your planning and the fact that your team is asked to deliver too much work. You could also determine from this data what teams work together productively and which don’t, which can be a key insight for future team allocations. Cycle time is another important metric used to measure the productivity of software engineering teams. It measures the time it takes for a team to make a change to a software system and deliver that change into production, which obviously holds value and can really give insight into the productivity of a specific team.

**Platforms and Infrastructure**

There are various platforms that companies use for analysing and reporting on the data related to the effectiveness of their software engineers and teams. In this section, I will go into detail about a few of them.

Pluralsight Flow is a popular platform that aggregates historical Git data into easy to understand insights and reports. A quote from their website: “With Flow, you can identify bottlenecks and compare sprints and releases over time. The result: previously uninformed conversations about engineering turned into debate-free, data-driven discussions”. With Flow, they provide a proficiency report which analyses the effectiveness of their software engineers individually. With the report, ‘Skill IQ’ data is gathered which identifies knowledge strengths and gaps of each software engineer. Skill development and workflow recommendations are then provided, and with Pluralsight having hundreds of courses on upskilling engineers on specific topics, these courses provide the engineers with the opportunity to fill in the gaps of their ‘Skill IQ’. Flow also provides an extremely informative work log. Work log acts as a superior replacement to stand-ups and meetings, providing a single view of a team’s contributions and habits. With work log, you can view a code’s commits and PRs in a single dashboard, while also being able to view project bottlenecks with the assemble data, which may have not been obvious beforehand. Flow also provides a feature called ‘retrospective’, which provides tools to evaluate the success of releases and compare sprints. Retrospective provides informative graphs such ‘codebase impact’ being plotted against time to show the companies most and least productive time periods. With the functionality that Flow provides to companies, they cite these statistics on their website: a 21% increase in coding days per week, a 25% increase in commits per day and a 20% increase in impact to the codebase.

Waydev is another popular platform that uses git analytics to measure software engineering. A quote from their website: “Align business initiatives with engineering work to establish unified goals and success metrics”. Waydev provides features to analyse project costs to provide insight into the progress and costs of key initiatives and deliverables to help teams ship on schedule. They also provide analysis of resource planning as to ensure that the engineers’ resources and team dynamics are optimised to improve software velocity delivery among each engineer. One of Waydev’s key features is the ability to generate custom reports to query more precise questions to the data. Companies can get answers to questions such as “How was the coding efforts of different software engineers affected by remote work?”, which can provide key insights into the varying personality and overall ability of the engineers which other tools wouldn’t pick up on. Waydev cites several statistics on their website of the increased performance of engineers following the integration of Waydev’s platform in a development environment. Examples include a 22% acceleration in time to market, a 28% increase in cycle time, a 21% decrease in unplanned work, a 15% increase in coding days per week and a 24% increase in features delivered.

Code Climate is the last example of a platform that measures software engineering that we will look at. Code Climate is a web-hosted software which provides organizations and corporations the ability to take charge of their code quality by incorporating fully configurable test coverage as well as data maintainability throughout the workflow development. Code Climate provides an open and extensible platform for ensuring code health which analyses billions of code lines every week. The software has been able to make the lives of developers easier by improving their Code Climate workflow.

As you know, codes can have some subtle issues which wouldn’t be obvious, which is why the software helps developers trap these issues by analysing each pull request before it is integrated. One of the features of Code Climate’s functionality is test coverage. Code Climate allows software engineers to test the coverage and range of new codes, providing them with insights of what that code can do as well as all the processes involved in it. It also provides them with per-file and overall coverage, meaning that the whole complex lines of codes are independently checked and in perfect working condition. The software allows engineers to pull up request statuses which can help them make decisions on their code. By using the software, engineers can track the progress of their code from week to week. The progress report they are provided with shows a summary of their most important changes which were merged. Engineers are also able to visualize long-term trends in the codebase. This can help them determine if the overall quality of their codes is getting any better or worse. They can also get a broad view of how their projects are faring next to others, and see which ones are lagging so they can pay them more attention.

**Computation and Algorithms**

Various methods of computation are used in the software engineering workspace to measure the productivity of engineers. In this section, we will discuss a few of them.

One example of the measuring of productivity in the workplace is the use of wearable technology to measure the happiness of individuals. Although this study was not carried out specifically with software engineers, I still believe it to be relevant. Workers wear a sensor on their chests while at work and this sensor does various computations in identifying the activity of the individual. The researchers identified characteristic patterns of physical activity that have a strong correlation with happiness. A very interesting idea produced by the study was the direct relationship between high physical activity, happiness and productivity. Of course, it is common sense that a worker that is more physically active throughout the day will be more productive but the study goes further and shows that workers who are physically active on their breaks through higher levels of conversation with colleagues etc show a higher level of happiness and in turn a higher level of productivity on that given day.

**Ethics**

The more development and progress in the topic of measurement of productivity in software engineering, the more prevalent the questions of ethics become. Given free roam, the measurement of productivity could become extremely personal and invasive to the point of measuring and recording every minute action performed by an individual throughout the day, maybe even outside of the workplace! With the technology of wearable technology that can measure productivity through physical activity already in place, it is inevitable that even more sophisticated technologies will be developed that can perform even more accurate computations on the actions of an individual. However, these technologies, with the invasive and private information that they will be able to collect, will have to adhere extreme ethical considerations with how they handle the data. With all the private information it will amass, there will have to be measures in place about what information the technology can and cannot collect, and what types of information are readable by the managers and bosses of the engineers being measured. Of course, if proper ethical procedure isn’t followed, there will be an inevitable push-back against the technologies by engineers and could even create a vast distrust in the use of technology for measuring productivity, which could be detrimental to this field of work.

So, although we have to be careful about what data is being collected and how it is being used, I do believe the measuring of software engineering productivity to be a noble pursuit. If done right, it can help everybody involved. This would mean higher productivity from individuals, which in turn could mean higher levels of happiness through accomplishment and this higher level of productivity obviously benefits the business.