CSU33012 Software Engineering

Renaldas Sapaitis

17324469

Measuring Software Engineering

"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."

Software measurement is a quantified attribute of a characteristic of a software product or the software process. manifestation of the size, quantity, amount or dimension of a particular attributes of a product or process. It is an authority within software engineering. Software measurement process is defined and governed by ISO Standard.

Software is measured to create and anticipate current and future qualities of the product or process, and to enhance and regulate the state of the project in relation to budget and schedule.

There are two types of software measurement:

1. **Direct Measurement:**
   In direct measurement the product, process or thing is measured directly using standard scale.
2. **Indirect Measurement:**
   In indirect measurement the quantity or quality to be measured is measured using related parameters i.e. by use of reference.

A software metric stands for a potential area where measurement can be effectively applied to a certain software module or its specifications. In other words, a metric assumes taking some data from your application development lifecycle and using it for measuring software developer productivity.

Measurable Factors

Code

Probably the easiest way to measure the amount of work done by a software engineer is to count the number of lines of code they have written. This data can be gathered through a version control system such as git or bitbucket.

Source lines of code (SLOC) is a software metric used to measure the size of a computer program by counting the number of lines in the text of the program's source code. It is usually used to predict the amount of effort that will be necessary to develop a program, as well as to estimate programming productivity or maintainability once the software is produced. A problem with this however is that measuring performance by the volume of code written does not take into account coding standards used, types of algorithms and databases if any. It is not the ideal way to measure software but coupled with other metrics, this could prove to be a useful measurement technique.

To combat the limitations of SLOC, one could measure the volume of code with logical lines of code (LLOC). A logical line of code is one that contains actual source code. An empty line or a comment line is not counted in LLOC. Even though these measure code differently, it is not easy to find a way to measure code fairly when looking at LLOC only.

Defects are an inevitability at some point during the development life cycle, particularly for larger and more complicated projects. Though some defects are a given throughout the course of development, production defects can often be avoided with intelligent planning, coding practices and thorough testing. The industry average defect rate is around 1-25 bugs for every 1,000 lines of code. This provides some insight into the quality of the software being written as well as the skill of the engineer writing it, as well as their skill level with the tools they are using.

Fixing bugs can often be a tedious task, but can be made easier if the original code written adheres to a good standard. This is vital in projects where the total amount of people working on a piece of software is in the thousands, and where the total lines of code are in the millions. Fixing bugs becomes second nature when the developers have a good understanding of the problem they are faced with, as well as how that piece of code interacts with other parts of the software being designed. The difficulty in measuring software here is that fixing a bug can take a long time, but the amount of lines changed may not be impressive. This does not mean that a large amount of time was spent fixing a small issue, quite the opposite. Even though the quantity of code produced might be low, it does not mean it is useless. It is hard to measure the performance of an engineer in a case like this, as it dependant on the difficulty of the problem at hand which is, again, difficult to measure

A decent way to measure the consistency of work produced is to consider the quantity of code produced with respect to time. Again this has its positives and negatives such as in gauging a developers performance over time, but may not consider the difficulty of the project at hand, or the teams or individuals strengths and weaknesses.

Code churn is a natural part of the development cycle. high code churn can mean lots of things have changed and need testing or reviewing. Ideally you see lower code churn the closer you get to a release date. Code churn is the percentage of a developers code that is an edit of their recent work, in other words, how much of their code is being rewritten. Several tools can measure code churn for you; Azure DevOps Server has an in-built mechanism for working it out, and SaaS offerings exist, such as GitPrime.

Commit frequency is another useful way to look at the software development process. The number of times a developer commits a day and their commit behaviour over time can be used to analyse their performance.

Pull requests and code review can also prove to be useful when determining the quality of work produced by a software engineer. Pull requests allow members in a team to comment on code written by others and can be a very important way to allocate

valuable resources within the team. A team can work within the confines of their strengths and weaknesses efficiently provided the team is allocated work based on what they are good at. It is necessary to do code review to improve the efficiency of a team.

Feedback is also necessary and can be useful to gauge how well a team works together. For example if the feedback is constructive it would imply the team functions well as opposed to critical feedback which would imply some interpersonal relationships would need to be worked on in the team. The employer could make use of this code review and feedback and determine what the best course of action would be in terms of allocating engineers with specific skill sets to different teams, who work on different aspects of a project. Not everyone is in the top 1% of software development, something that Fred Brooks touches upon multiple times in his “No Silver Bullet” paper.

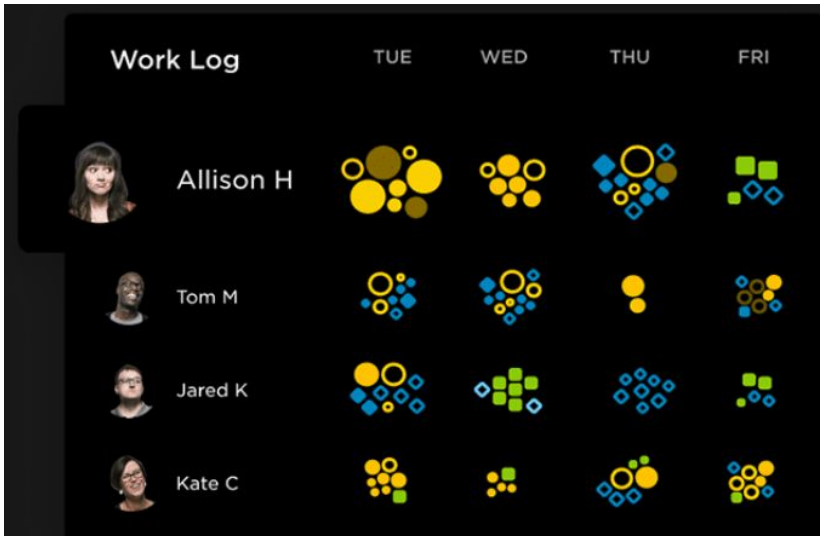
Computational Platforms

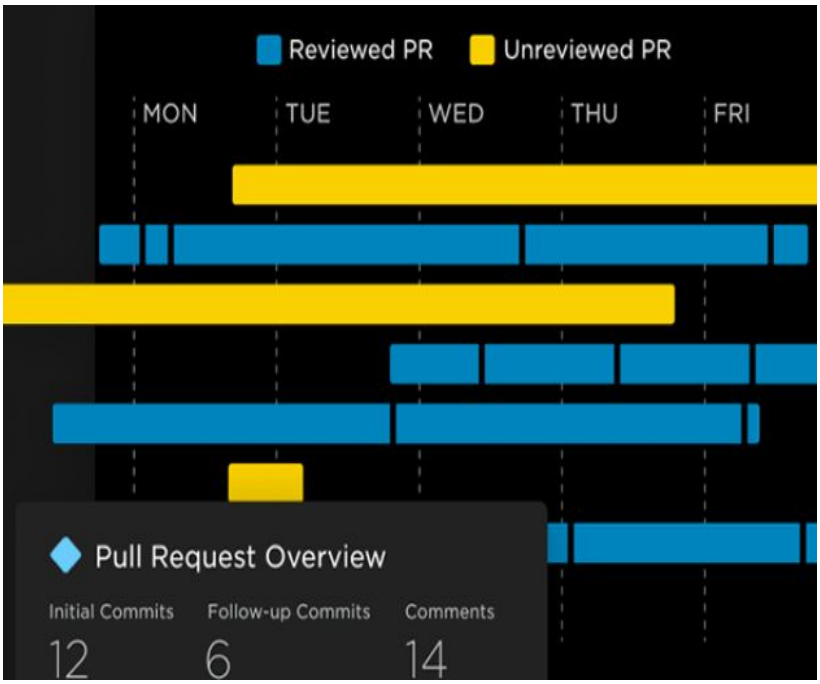
A computing platform is the environment in which a piece of software is executed. It may be the hardware or the operating system as long as the program code is executed with it. There are a large number of computational platforms that aid in measurement and analysis of software engineers and engineering processes.

Pluralsight Flow provides workflow analytics for software development teams. By analyzing metadata from git, code reviews, and issue trackers they help engineering teams move faster, increase effectiveness, and debug their development process with data.

With Flow, you can:

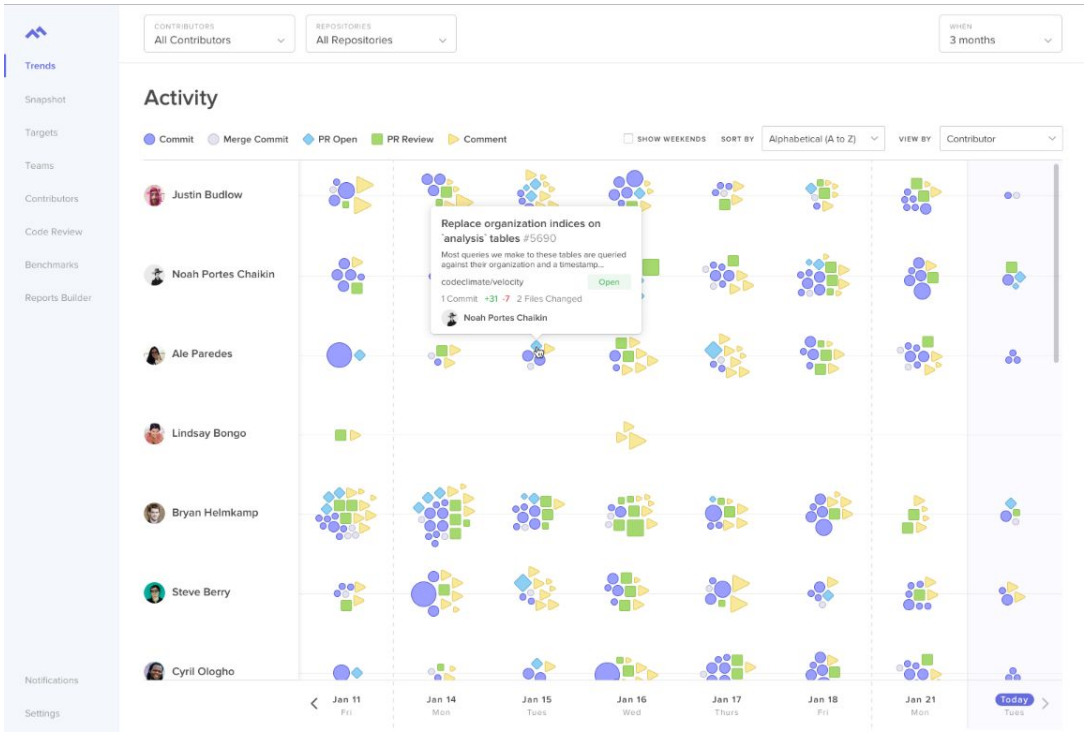
- See how much time is spent refactoring legacy code against new work
- Recognize project bottlenecks and remove them
- Get concrete datta around commit risk, code churn and impact to better guide discussions.





Flow also provides a powerful visualization of your team's code review dynamics. It allows you to quickly spot unreviewed and old PRs or lengthy discussions in comments, and measure how collaborative a group is and track how these trends change over time. This is without a doubt a very useful way to measure software engineers.

Another computational platform commonly used is Velocity. Velocity is a product made by the company Code Climate. The difference between Velocity and Flow is that unlike Flow, Velocity is aimed at finding process constraints in departments, not analysing individual developers.

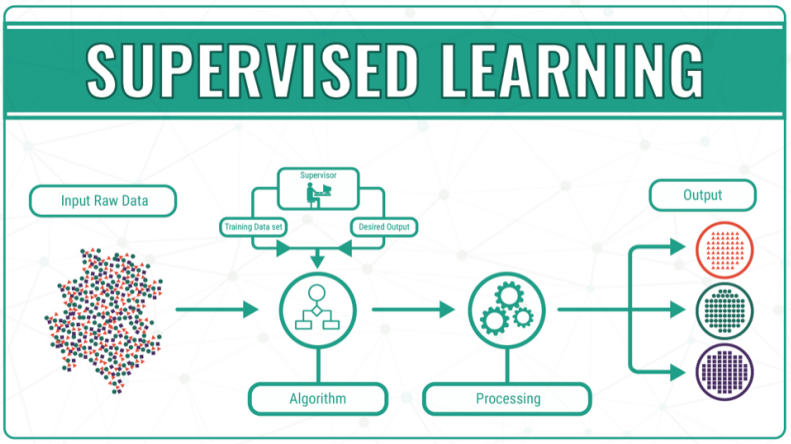


The story of its origin is quite friendly. When Chris first stepped into his role, he noticed tension among the team. Through 1:1s and conversations with prior leadership, he uncovered a culture that was hostile to mistakes. Chris says, “When I first started, errors were a source of shame. Everyone was afraid of making a mistake because they didn’t want to be belittled.”, “We made it clear that we expect failures on this team. I have them, team leaders have them. What’s important is how we handle the recovery process.”

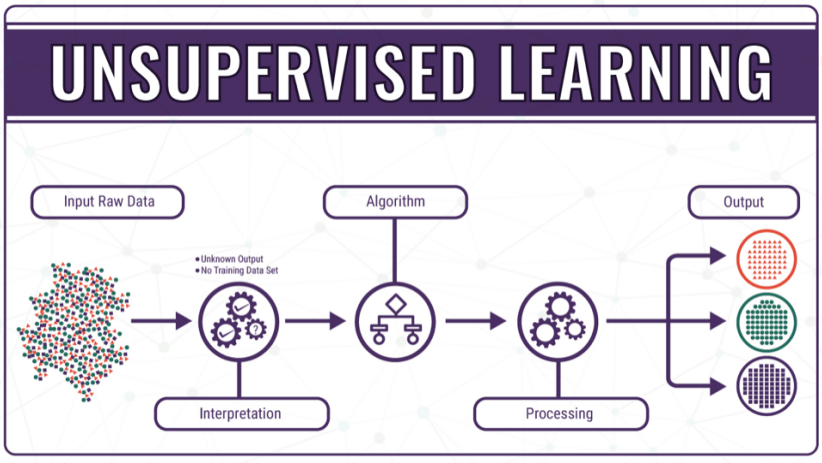
Algorithmic approaches

Artificial intelligence and machine learning are increasingly being used to analyse large amounts of data. These methods have proven extremely useful in solving problems such as translation, image recognition and self driving cars. Three basic machine learning paradigms are outlined below.

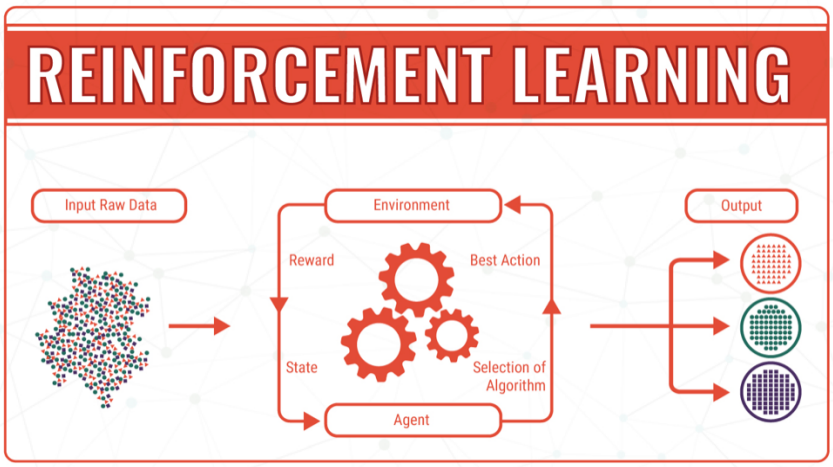
Supervised learning is the machine learning task of learning a function that maps an input to an output based on example input-output pairs. It infers a function from labeled training data consisting of a set of training examples



Unsupervised learning is a type of machine learning that looks for previously undetected patterns in a data set with no pre-existing labels and with a minimum of human supervision.



Reinforcement learning is an area of machine learning concerned with how software agents ought to take actions in an environment in order to maximize the notion of cumulative reward.



Ethical concerns

Ethics is a research field that is obtaining more and more attention in Computer Science due to the proliferation of artificial intelligence software, machine learning algorithms and so on. Ethics research has produced till now a set of guidelines, such as ethical codes, to be followed by people involved in Computer Science. However, little effort has been spent for producing formal requirements to be included in the design process of software able to act ethically with users.

I believe there is an ever looming concern that even the aforementioned software, will one day track not just the lines of code a developer writes, but the time they take to set up their work environment in the mornings, the time they take to eat lunch, what they have for lunch, when and how they exercise based off their heart rates that may or may not be monitored as mentioned above. The time they take to go to and from work. The feeling of constantly being watched can have an adverse impact on the mental health, and general health.

I think it is important to have a dedicated work time too, and anything after the allocated time you are given should not be used for work purposes. It is not an uncommon sight to see people, not just software engineers but teachers, lecturers, researchers, doctors devoting their time to work and not getting paid for it directly, for example grading work done by students after college hours. At what point does it become too much? when a developer has to work into the early hours of the morning every day of the week just to get a piece of software finished that meets spec?

Gathering data based on code is of little concern however, as this is data the software engineer gives the company anyway, as they have to finish a product or service within spec and deadline. On the other hand though, it can be manipulated in such a way that makes the developer appear like they have done more than they actually have, in other words, to appear more productive.

In class lectures, we had a discussion about the ethical implications of measuring software engineering, and one of the points brought up was the measuring of personal and health data. This can have its uses, however constant monitoring of heart rate, beats per minute, blood pressure etc. provides too much information about a person, let alone an employee. Think about an employer using this information to control an employees personal life in terms of exercise, sleep, diet.

This would naturally lead to other personal information being gathered such as recording of employee voice data, conversations, or monitoring of social media which I believe is not ethical. This type of information gathering is far too intrusive and could pressure an employee to be in a constant state of professionalism, even when not at work.

Conclusion

There are two sides to the coin when it comes to measuring software engineering. On one hand measuring software engineers can undoubtedly provide a charming work

environment where employees are rewarded fairly based on their work. It can be used to identify and solve problems before they are too big to deal with, and it can reward those who flourish in a team based environment. This does all depend on the methods used to gather this data and no doubt, the type of data gathered.

On the other side of the coin, gathering certain data like the aforementioned health and personal data may have unethical implications that could drive potential employees away from a particular workplace, where the employers can have too much controlling power that they can use. Based on work ethic they have the power to allocate resources as needed, but can also remove unwanted employees from the team, something that can be fair if an employee is legitimately more of a nuisance than an asset, but can also be unfair if a developer unexpectedly becomes sick and cannot perform to the best of their abilities. As with all things, there must be a good balance, especially when it comes to ethical data gathering and storing.

Sources:

- https://www.ukessays.com/essays/information-technology/importance-of-software-measurement-and-metrics-information-technology-essay.php
- https://en.wikipedia.org/wiki/Software_measurement
- https://www.geeksforgeeks.org/software-measurement-and-metrics/
- https://medium.com/@infopulseglobal_9037/top-10-software-development-metrics-to-measure-productivity-bcc9051c4615
- https://www.aivosto.com/project/help/pm-loc.html
- https://www.infopulse.com/blog/top-10-software-development-metrics-to-measure-productivity/
- https://airbrake.io/blog/devops/production-defects-are-not-inevitable
- https://textexpander.com/blog/what-is-code-churn-and-how-to-reduce-it/
- https://www.pluralsight.com/authors/git-prime
- https://codeclimate.com/

- https://www.researchgate.net/publication/342474767_Measurement_of_Ethical_Issues_in_Software_Products
- https://towardsdatascience.com/supervised-vs-unsupervised-learning-14f68e32ea8d?gi=4755f376628
- https://codeclimate.com/