CSU33012 Measuring Software Engineering

Anton Yamkovoy – 17331565

Introduction:

Today many aspects of software engineering desperately need measurement and looking after, due to methodologies such as Agile Development, Scrum, these methods of rapid prototyping and often short deadlines for parts of projects come hand in hand with the management responsibilities and the need for companies to measure the progress of teams and individual engineers with sometimes seemingly absurd metrics that many people would consider needless or inconsequential.

With continuous integration and testing of newly implemented features and rapidly newer versions coming out of some companies daily or weekly, the question is whether there is significant improvement in productivity by measuring teams and individuals using existing metrics, to ensure high productivity, the contentedness of their teams, and to make sure work is distributed evenly.

Measurement and Gathering of Metrics:

There are many existing metrics that are used to measure the output of a software engineer or engineering team, Metrics are often made to some ones personal preference, But without metrics often businesses or companies cannot be successful, to many people this isn't pleasant to hear, but mostly it's true, for some metrics are sales per month, for some completing a task before a deadline, for some hours worked.

The first type of metrics in my opinion are the metrics that are based of off the resulting product. Universal formulae for success in most things don't ever exist, usually the best option is assessed from situational parameters and the environment.

For most projects there exists and budget, and engineers are payed on a time based metric, however due to the vagueness of software engineering specifications it is sometimes impossible to predict how much time / effort something will take. In the freelance community a solution to this is to not have hard deadlines, where due to mistakes on timing a client may have to pay x2-10 times more, usually an agreement is made that the work will be done to a standard that satisfies the client in a suitable timeline which is flexible.

The second type of metrics are based off of time spent, but using these kinds of metrics the complexity grows very quickly, when talking about large projects with multiple teams and divisions. It is hard to measure the time needed to complete a project due to these factors, sometimes many teams working on a project, each

team in their own specialities has many workers, and the time to finish is often vague.

However a strongly connected relationship to hours worked is almost never going to work, you need other metrics, however some metrics are more useful than others, for example lines of code is the most simple metric that you can probably think of, but even if a developer writes a thousand lines of code on a single unit of time, without further analysis there is no way of finding out if that code is useful, bug free or that it benefits the code base, due to these problems it would be absurd for example to pay per line of code written in general.

With some small variations simple metrics can become as a base quite useful for managers

Time to complete a section of a project or the project, from specification to delivery to the customer is a good indication over many projects of how fast a team can complete an adequate job on a project of a given size, This metric can be transferred to similar size or type projects in the future given that the composition of a team hasn't changed drastically.

Another metric to see the general productivity of developers combined with other analysis which is useful is their activity on different working days which can be monitored through the version control system used by the company, The tasks that go into the "activity" of a day is writing code, solving issues, and completing code review, these activities can also be measured by their importance, but it is still hard to attain information about how important the code that was written was, or if the issues took much time to solve, or their difficulty level.

A metric that can be used to give some more value to activity of a given day or improve the "lines of code" metric, is the impact of a commit or code added, lower impact may only affect a single file in the codebase, while a high impact commit may have multiple instances of deletions / additions that affect multiple parts of the project and their reach is over many files.

Some metrics that are often used combine other metrics together to improve their usefulness, an Efficiency metric can be shown to be, the percentage of the engineer's code that is useful, which is usually done by comparing the code to it's longevity and it's review statistics, and with not many deletions being made to the code, the code staying mainly unchanged being the second factor, this provides us with quadrants of analysis, where the best code is mostly unchanged and has a high throughput, and other quadrants can show perfectionism, with low throughput and low amount of changes, or developers who's code receives a lot of changes but is also impactful.

Many other metrics exist, maybe useful or maybe not, in fact it is possible to measure many different aspects of software engineering as it is with many work fields, interesting ideas of measurement include the posture of engineers, the amount of impact at different times of day, even going to measure heartrate and other health indicators of engineers, to pursue ideal productivity, The problem is it is possible to measure almost anything, but to know whether these metrics are of any

use that is the hard problem it is hard to tell whether a new innovative metric will provide any real improvement, but these changes will come through research and experimentation, I would say that there are many unclaimed metrics that no one is using right now, that may turn out to be very influential in the future, but many would agree that there will never exist a truly universal simplistic measure to record the output of a project.

Computational Platforms Available:

The vast majority of programmers and teams use some kind of version control whether it be git / github, perforce or subversion and many others, These version control systems, especially with a powerful api such as github's api for retrieving all data logged by a certain user, you find yourself with a large amount of data to be processed which in some way shows your development team and process, Many tools exist based on version control systems' data to analyse the software development process, individual engineers and their interactions and own work.

One of the most famous platforms available in this field are gitprime and velocity being quite similar in principle. Both platforms have quite similar features:

When talking about data sources, Velocity began as a company with a focus on pull requests, it pulls data from git and git applications like bitbucket and github, code level data and collaborative data, such as reviews are pulled in, An analysis is performed using a data enrichment algorithmic technique and metrics such as "Impact", "Rework", "Pull Request Activity Level" and "Review Cycles" are formed, Impact is a measure that takes into account location of changes, the size and the nature of a change,whether it was a new implementation or just some refactoring. Rework shows that quality of the code that is being produced, the metric is updated when a contributor reworks the code that they have written themselves, A high level of rework can show that an engineer is having difficulties with a certain feature due to issues with the codebase knowledge or changing requirements in the project.

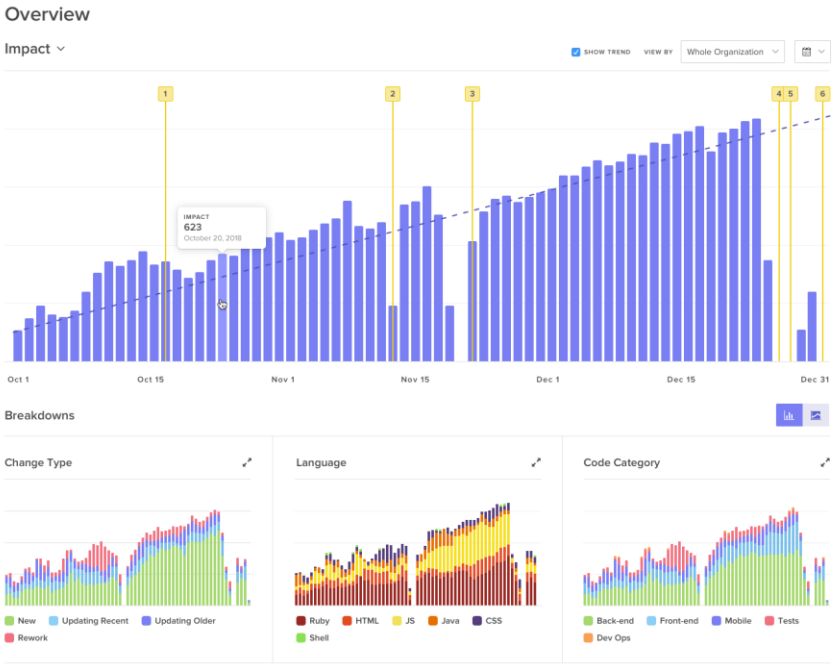
Pull Request Activity is a metric that estimates, the activity level of a pull request, it also provides a dashboard for managers to manage the current pull requests, A manager can gain insight into what problems are troubling the team at the current moment.

In comparison GitPrime's metrics, are computed using a proprietary algorithm

Impact, this metric estimates the difficulty of a change in the codebase, very similar to velocities mentioned above and in the measurement section. Their algorithm is based on insertion points, lines of code, and number of files changed.

Churn, is a metric that measures code that has been reworked, similar to rework above, It can represent wasted effort by individual developers, and can be used in combination with other statistics to measure other metrics.

TT100 Raw, is a metric used by GitPrime to measure the time taken to write 100 lines of code, not taking into account any quality constraints. Similarly TT100 Productive, is similar but takes into account churn and efficiency.



Another type of computational platform that exists is automatic code quality checking platforms, these include companies such as codeclimate, houndci.com and coveralls.io

CodeClimate which also provides velocity as seen previously and houndci, are tools for analysing code quality.

They form breakdowns of a given repository, highlighting issues such as

Issues in the code, this metric displays the number of issues that are found in the default branch of the project, Test Coverage, which shows coverage of tests on the default branch, CodeClimate also provides statistics for the latest build and refresh button for refreshing analysis on a project.

These metrics can be easily displayed for others on the repo using the badges feature on these sites.

Coveralls.io is a similar concept with more focus on code coverage and that new code written is not left out of testing and that it has test adequately written for it

Other useful tools exist for searching through code, such as sourcegraph.com,

It is a tool for open source development, it provides a search engine and code browser that is used to help developers find better code and build software faster,

It semantically indexes all open source code on the web, letting you search by repo package or function, linking docs, allowing you to jump to definitions, and find usage examples, performed in your web browser.

Aside from tools that compare metrics and allow for search functionality, there exists platforms such as Jira, these are used in combination with agile / scrum methodologies, to improve productivity not through statistics based metrics but team coordination and communicative improvements.

For agile development Jira is used for, issue tracking, scrum boards, estimation and work logging, workflow logging and manipulation, and other things such as project backlogs.

Jira also has functionality for management teams and pure software development teams which include more features for sprint planning and developer tool integration.

Together with Jira, many teams use a platform such as slack, to improve communication, instant messaging services are very common for almost anyone, however a work based platform can bring improvements also.

The number of similar platforms is growing year by year, and the problem is again mostly, which metrics have a real impact and have a positive impact on the coherence of work, and their impact on teams and individuals.

Approaches:

Algorithmic:

In the past productivity was often measured in simple metrics, that don't hold up in reality today, such as a popular metric historically

Productivity equals effective source lines of code written per person month.

In todays development environment in some places the ownership of code as a concept may be under scrutiny, so these kind of metrics, together with using scripts found elsewhere or collaborative work between developers shows that these kind of metrics are sometimes not as useful.

Today many companies and research teams are interested in automatic code quality assurance which goes past statistical analysis, and uses technologies such as machine learning and neural networks.

A neural network must be trained to recognise "good" or "bad" code quality with a combination of analysis of public code datasets which are plentiful using tools mentioned above in the platforms available section, together with manual training of this net, and some static analysis using Bayesian Classification, n-gram, support vector machine and decision tree, A system can be taught to identify snippets of code that aren't suitable or are of top quality.

Another interesting concept that is well known is evolutionary algorithms, which can be given tighter constraints and conditions to continue changing or "evolving" into a system to identifies some desirable characteristics in software systems or codebases. The system must be guided into a correct path, but this is a similar property of many of these machine learning / ai systems. They mostly require a manual initial learning process.

When talking about many applications in the real world, Boolean logic doesn't apply as nicely as it does in logical electronic systems, A research topic known as fuzzy logic may be used to analyse these kind of systems, It can be used to determine value added from work, in comparison to non-value added as well as potential productivity improvements for the most cost efficient way to perform tasks, It also has potential to develop assessments of current workflow processes and current operations, and provide recommendations for improved utilization and productivity.

Often it is difficult to measure, quality of code or team performance purely through statistics and probabilistic measures, It is useful to be able to train a system which retrieves information through, natural language, and outputs of code review and feedback to different engineers, with natural language processing technologies emerging. Some think that monitoring conversations or reviewing feedback on code written by people using a NLP system could bring more useful feedback and learning potential to a system which would be trained incrementally to give feedback on it's own based on a plethora of previous input by real engineers.

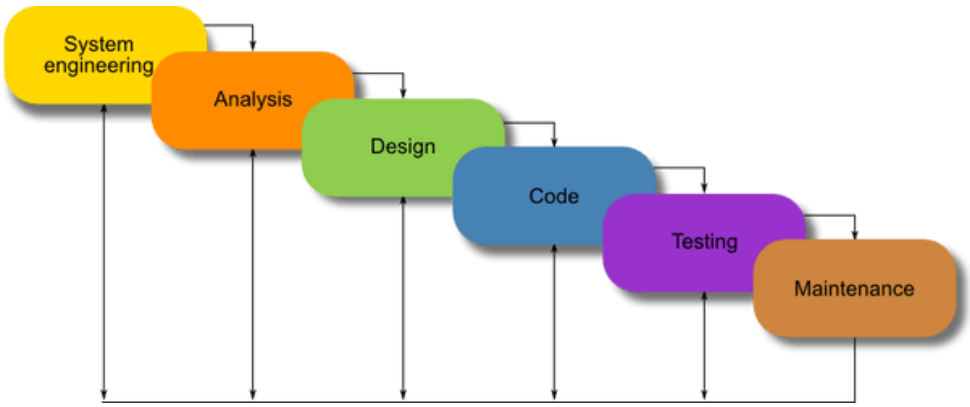
Another idea in a similar sense would be to measure the "mood" and "content" of slack or other messaging platforms for teams to compare statistics on different factors for example volume of messages compared to volume or work, or "moods" of messages compared to team satisfaction.

Methodology Based:

Many different methodologies have existed throughout time to efficiently approach the task of completing a software engineering project, The most basic being initially meeting with a client, writing up specifications and working on the project with a time based deadline given initially by the client.

Often this method leads to unwanted or not correct features from the view of the client, due to the difficulty of communication between clients and developers and often an occurring situation is that a project very often goes over time / budget or fails completely.

A more developed version of this initial method, is called “Waterfall”, It is also a linear methodology, consisting of stages with succeed each other, the first of which is setting up requirements for a project, this consists of meeting with a client or their representatives and drawing up requirements documents, for functional and non functional requirements, The next stage is the design stage, where models and prototypes are made before starting the implementation phase, where the proposed model is implemented in a framework/language that is sometimes not specified by clients, The next stage is verification where the product is tested and eventually released to the client, and reviewed so that it still fits their constraints, similar problems can occur in this stage, if the assumption is made that clients haven’t been making changes to the requirements throughout the development process, Waterfall doesn’t take this in to account. The last stage of development is often the longest and most time consuming, depending on the agreement made with the client, this is the maintenance stage, often bugs are found within the system after everyone has finished developing, and maintenance and updates are needed regularly, especially if the product has a large user base.



The linear nature of waterfall makes it easy to understand and easier to execute than other methods, however, progress is often slow and costly due to the rigid structure, this often leads developers to explore other methodologies.

Namely agile development. Agile is a methodology which minimizes risk when adding new features or functionality, In all agile development teams, the software is developed in micro iterations, short cycle iterations which contain small additions of further functionality, agile includes into itself sometimes scrum, extreme programming, and feature driven development.

In agile requirements evolve as the project goes on in comparison to some more traditional methods, Scrum in a management methodology contained inside agile, it provides a process framework, associated with increase in quality of deliverables, coping betting with changes in requirements and expecting those changes, and more control and better estimations of project times and costs.

In scrum, a sprint is set typically a short length of time less than a month, to complete some certain tasks, meetings are held in the team every day, and developers talk to everyone and give a presentation of what they are currently working on, whether they have any problems, these meetings and constraints are enforced by the scrum master, Often the scrum master isn't a fully fledged engineer, they provide the guidance for the scrum process itself, not necessarily technical details.

The tasks inside a sprint are spread out between developers, the daily meetings that occur are also a potential way of measuring progress quite frequently if you are trying to concur some metrics out of the agile development process.

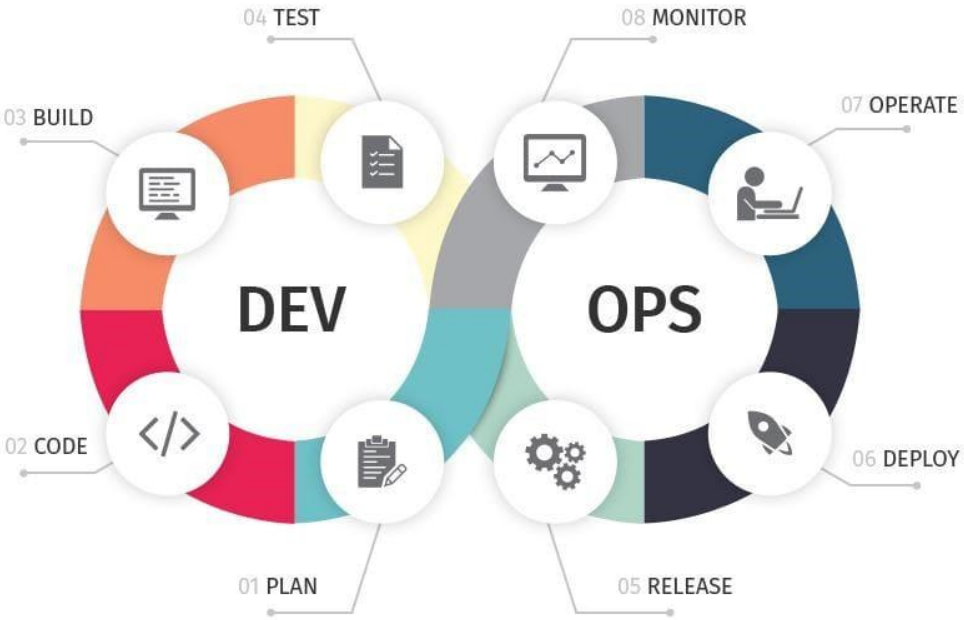
Another recently more adopted methodology is DevOps, the principle is that the focus is transferred from bursts of "releases" and defined releases for updates and product changes, in DevOps a method of "continuous development" is used, There is a focus on improving time to deliver to the client, shortening time before any bugs are fixed and allowing for minimal disruption, this is seen very clearly in large

services which will cause problems and negative attention if they are offline for significant time, for example any large web service or social media site.

DevOps is usually beneficial to the product and development process, however it isn't suited to being measured by metrics for example in comparison to agile's daily update meetings and platitude of information released about the development process, This can case a problem for less organised or less experienced teams who may need to rely on such stats.

A useful metric in DevOps to combat these kind of problems, are batch sizes, these are the average amount of backlog items in any stage of delivery. Batch sizes can be used to predict the length of release cycles and predict future performance, Together with other statistical metrics, it can be found out why batch size is growing at a rate which may be unsustainable to the progress of the project.

There exist many other methodologies, however the point is that these processes and descriptions of workflows are crucial in the improving productivity that has been seen over the last decades in software engineering, they are often more important than metrics based approaches in improving productivity of individual engineers and a team as a whole in the process of completing projects.



Ethical Concerns around Software Engineering Analytics:

Most of the measurement methods and methodologies would likely bring substantial improvements, to a teams productivity, however many issues arise due to privacy concerns and ethical concerns, you don't even know will the improvements brought in by monitoring your employees be offset by a loss in "morale" or a factor of lost performance due to higher stress, similar to an interview or exam situation, it is very clear that the majority of people have a hard time performing sometimes simple tasks when there is a high stress situation or constant over the shoulder monitoring systems.

Many large tech companies have tried to balance these problems out by providing other workplace factors that contribute to increased comfort, and provide an incentive for workers to stay at work, such as in building services like a gym, hairdresser or food served at work, I would say that the majority of people don't have a problem with these types of strategies, however the companies wouldn't be doing them if they weren't leading to more time spent at work, in many cases working overtime and having lunch breaks partially removed, this could be considered sly behaviour.

When considering physical monitoring devices, and extending to scanner cards to monitor your location in the building on the light side, to full on implants in a workers body for "ease of access", Many people would have hard objections to such devices being used, Aside from privacy problems regarding to such monitoring devices, having electromagnetic devices inside your body will almost certainly lead to heath effects in the future, many people would prefer not to have their health statistics measured, such as heart rate, activity, posture, which, could lead to unjust decisions regarding the future of a employee at a company, a obvious example being a pregnant woman coming up to a promotion choice.

Another side to this issue, would be the mass collection to data on employees, and as with most information gathered right now in different industries, the data is usually sold to the highest bidder, This would lead to recruitment based on statistics gathered at work, with the problem being whether these stats are a good representation of the individual.

When basing a grading system of in reality quite easily manipulatable data, it is nearly always possible to find out how the statistics are being performed, and by focusing on certain factors, being able to improve the engineers rating, without in reality doing quality work, assuming many people in a company are taking on this approach, the work becomes second priority, and the first priority becomes out playing the detection system, leading to good ratings all around, and work not being done to the best standard, a highly competitive workplace, with not much incentive for real work and collaboration leads to an unpleasant and toxic workplace, as software engineering is primarily a team process, similar more to team sport, in comparison to sports which are only on a individual basis, where such rating systems would be more suitable.

The morale and cohesion of a team is often more important than the individual composition. When talking about a system where the bottom x percent are removed every year or a cycle or arbitrary length, you will end up with engineers who perform well by your metrics, however a “Rockstar” engineer, will often get bored by mundane and necessary tasks, that need to be done for the project to continue, if you remove all of the people from the system who may be able to work consistently but not to the desired level, your top level staff may be left with this work for themselves, and due to ego problems, and their overqualifications, they may not want to work in this project anymore.

There is much to be desired from the methods that we gather data using, and there are many pitfalls, especially due to ethical and environmental reasons in a team, it would be foolish to implement these systems and expect the mood and cohesion of a team to stay unchanged, since the environment was changed drastically, not to say that these problems aren’t to be overcome, but I don’t think that measurement and surveillance and the be all and end all of software engineering.

Conclusion:

In conclusion, the field of software engineering, is very complex, and hard to measure on a level of individuals or teams, I don’t think there is a way right now of defining productivity very well, as the way we work is changing at a pace that is impossible to keep up with for the measurement techniques used right now.

However as we gradually see the new introductions of new methodology based approaches and more sophisticated statistics based measurement, these will provide more interesting insights into the path forward, the methodologies also provide a way to go further away from scrutinising metrics and lead to a more team based strategic approach.

In the real world it is hard to talk about these kind of ideas, since many companies especially smaller companies simply don’t have the organisation or experience to in reality implement many of the ideas described above.

Bibliography:

Jira Software features - https://www.atlassian.com/software/jira/features

Methodologies - https://www.researchgate.net/publication/255710396_Software_Development_Methodologies

Gitprime and Velocity resources - https://www.gitprime.com/resources/ , https://codeclimate.com/velocity/understand-diagnose/

Software engineering methodology productivity - https://ieeexplore.ieee.org/document/237006

Algorithmic methods - https://semmle.com/assets/papers/measuring-software-development.pdf

DevOps performance - http://devopsenterprise.io/media/DOES_forum_metrics_102015.pdf

Agile development - https://www.pmi.org/learning/library/agile-project-management-scrum-6269

Ethics for software engineers - https://sites.google.com/site/advancedsofteng/software-engineering-as-a-profession/codes-of-ethics-for-software-engineers