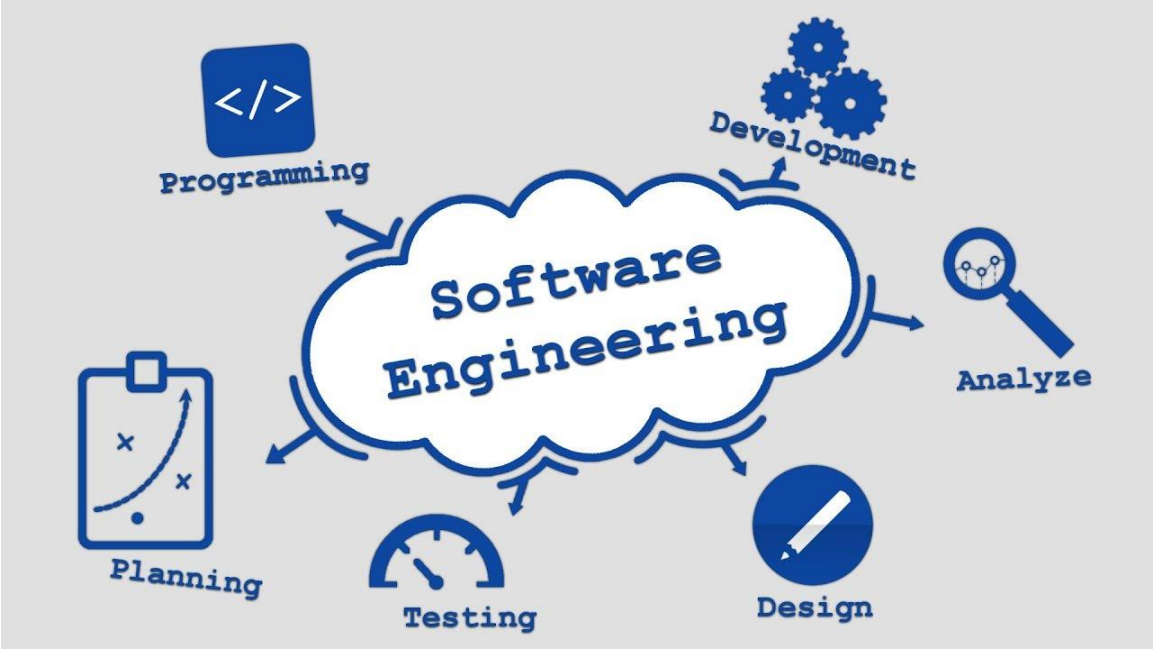


Measuring Software Engineering

A REPORT ON THE ANALYTICS OF MEASURING SOFTWARE ENGINNERING

Gabriel Barat 16327506 | Software Engineering CS3012 | 2018

Introduction

Software Engineering is the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software" in its most basic form. So to the normal every day individual or even some programmers themselves measuring software engineering may seem like an easy task, that will follow the same principles as a normal performance measure of other jobs. I, myself while studying Computer Science and Business work part time as a sales person to gather some experience in the field and generally found that jobs here are measured by KPIs (Key Performance Indicators). These KPIs generally just measure raw sales figures such as total sales and how much attachments individuals and the business as a whole sell to measure and compare against planned figures. So when I came upon this topic of measuring software engineering in terms of actual data and providing some platforms of how to do this including how ethical they were I was a bit puzzled as I thought there's really only performance numbers attached to this even though it's a totally different field. But as I investigated further, I have found many different levels of data that can be obtained, different views and also some ethical issues that come with some of these.

Software Engineering and its Measurement

As stated above Software Engineering is the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software" in its most basic form. However the measurement of Software Engineering can be described as gauging how well a piece of software is being developed (or how well it was developed) using some kind of measurable metrics such as simply counting how many bugs exist within the code.

Most individuals and firms engage in the measurement of software for the following reasons :

- To get an idea of the quality and progress of the current project at hand.
- To predict where this project is going in the future.
- To improve the quality of the current project.
- To get an idea of where the current project lies within the bounds of the groups budget and schedule.

All of this proves to be very productive to both managers and the engineers/programmers themselves. For managers this process can provide valuable info on how costly the project is, how productive the staff and their code is, will the customer be satisfied and how can the project improve. For the engineers/programmers on the other hand info on whether all bugs & faults have been fixed, have all goals been met and what can be added, can be obtained through the process. But how come all this was needed? Companies in the last half of the 20th Century started to realize how inefficient some of their software was which was beginning to have a costly effect on them due long schedules, major cost overturn, low quality & poor user satisfaction and thus started to pay more attention to their software measurement. Most of this involved more thoroughly comparing results to goals and adopting a less isolationist approach to the metrics used in the process which in turn provided more useful information on how to improve.

Measurable Data and Metrics

When it comes to measuring the software engineering process we can use different types of data to provide us with useful metrics. The team/project/company should try to determine the best data to use by keeping them within some scopes. Some of these scopes could include cost & effort estimation, productivity measures/models, quality measures/models, reliability models, performance models structural metrics and evaluation of methods.tools.

SOURCE BYTE SIZE

Source Byte Size is a measure of the actual size of the source code data and measures the file size versus packages, classes, methods etc. Overall source byte size of a project can be estimated which would then give a better idea of what kind of hardware can run the program. This could be important on devices of limited computing nature such as washing machines, dishwashers, smart fridges. Furthermore from this source byte size can also vary greatly from programming language to programming language. For example a program developed in Java could be smaller than the same program developed in COBOL. Even things such as executable byte size of programs written in different languages and different compilers can result in different sizes. As more development happens overall source byte size increases and thus more storage space is needed to store files.

Nowadays due to storage not being as much of an issue as it is use to be in the past source byte size is becoming a less important measurable data however it can still be used to enforce some company standards. For example if the program needed to work inside a limited machine like a washing machine and thus actually work at all source byte size would be important. Also if the company has any goals when it comes to indexing source byte size would be a good indicator as indexing takes slightly longer every time the source byte size increases.

SOURCE LINES OF CODE (SLOC)

This measurable data gives the amount of lines of code present in the program as the name implies. This data can be used to measure effort before as an estimate & after as an actual value and also give developers an idea of the size of the project they are working on. However source lines of code are not as meaningful as other metrics by themselves as quantity of code does not always amount to good work if those lines of code are of no use or add no value. So other metrics need to be added into the picture to give more meaning to this data. For example while taking the source lines of code, maybe adding the ratio of good code to buggy code will give an idea of quality as well. This would be important if the company or group has a focus on producing quality products instead of just producing products fast.

However source lines of code can also be measured in a different way by looking at how many lines of code are written in a certain amount of time. This is useful when looking at the performance of individuals. But when doing this a few things have to be taken into consideration such as which code should be taken into account such as auto filled in code by the editor. In this regard we can have two situations.

1. Physical SLOC = Lines of Source Code + Comments + Blank lines
2. Logical SLOC = Number of lines of functional code (“statements”)

This would be important for firms if they are looking to get a certain amount of work out of individuals when working on projects with strict deadlines.

TEST COVERAGE

Test coverage is a measurable data which shows how much of the code written in a program has been tested through various means, unit testing being a popular one. It is generally very popular and is taught as one of the better ways to test your code in college. Generally test coverage is given by (number of lines tested /total

number of lines) x 100. There are a few problems with test coverage however. Some people argue that sometimes the tests written in test coverage are a bit meaningless as people may just write tests that call certain lines so the test coverage goes up. This is a fair argument however if paired with proper quality control, test coverage can be a good measurable data to make sure code is running as desired without bugs meaning a better quality product.

HAPPINESS

To break away from some of the more traditional measurable data of software engineering there has been some research done on whether happiness can improve someone's work with some people going as far as using wearable tech to measure happiness and see the correlation between happiness and working better. Some other research shows that working with a better team that is trained and dependable can also boost productivity due to a better atmosphere. Overall some researchers seem to think that happy workers work over 30% better. However happiness can be very hard to measure as it is not a physical thing that you can measure but instead a feeling with no scale and therefore hard to quantify. But nevertheless companies such as Google seem to put a big emphasis on it where various great works are produced, so if productivity is a goal of the firm maybe happiness may be a good metric to look into.

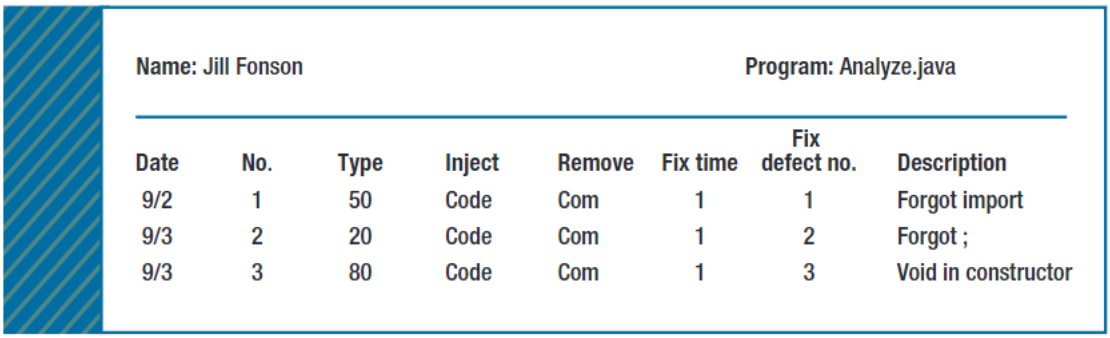
Computational Platforms

PSP- PERSONAL SOFTWARE PROCESS

PSP is a way of gathering data about the work of an individual on a piece of code or project. Generally a lot of information is collected in this process such as a project plan summary, time recording log, defect recording log, process improvement proposal, size estimation template, time estimation template, design checklist and a code checklist.

All of this information is then analyzed and compared to the scheduled plan and notes any differences. This first of all helps to track the performance of the engineers/programmers for themselves and also the manager. Secondly however it also helps determine if any of the differences were due to any distractions or environmental issues and helps to correlate them so as to find the problem. This then should help with productivity as we can now get to the root of the problem as we have proof that it affects performance.

However there are a few problems with PSP. To start off the whole platform takes a lot of work with many forms needed to be filled in manually which consumes a lot of time that could be used to be doing something else. Furthermore the platform is also prone to error as the forms are filled in manually so human error is a big risk here. Due to these reasons PSP is sometimes shyed away from.



Name: Jill Fonson — Program: Analyze.java

Date	No.	Type	Inject	Remove	Fix time	Fix defect no.	Description
9/2	1	50	Code	Com	1	1	Forgot import
9/3	2	20	Code	Com	1	2	Forgot ;
9/3	3	80	Code	Com	1	3	Void in constructor

LEAP TOOLKIT

The Leap Toolkit is suppose to be an improvement on the PSP platform. It seeks to improve on the shortcoming of PSP by reducing the amount of work needed to be done by the developer or as others would put it “reducing overheads”. It reduces the work needed by automating and normalizing data analysis of the data still inputted by the user.

However some people sometimes describe Leap vs PSP as Campfire vs Candle Light. The reason for this is that Leap is a lot more powerful but to use it you must go to the “Campfire” which means getting some data is quite easy but other data not so much. PSP the candle light on the other hand can be “moved around” so some data is more accessible. Overall though Leap does improve on PSP by reducing the overheads of people using the platform.

HACKYSTAT

Hackystat could be considered the second upgrade to PSP or the third variant in the family. It seeks to improve on the faults of the Leap platform as Leap did for PSP. Hackystat does this by not only reducing the overheads of the person using the platform but by also eliminating the input of the data manually which was very time consuming. Instead of a person inputting the data now the system directly takes the data from the tools used to develop the software through its own means.

Hackystat is split into 4 features:

- Both client and server side collection.
- Unobtrusive data collection.
- Fine grained data collection.
- Both personal and group based development.

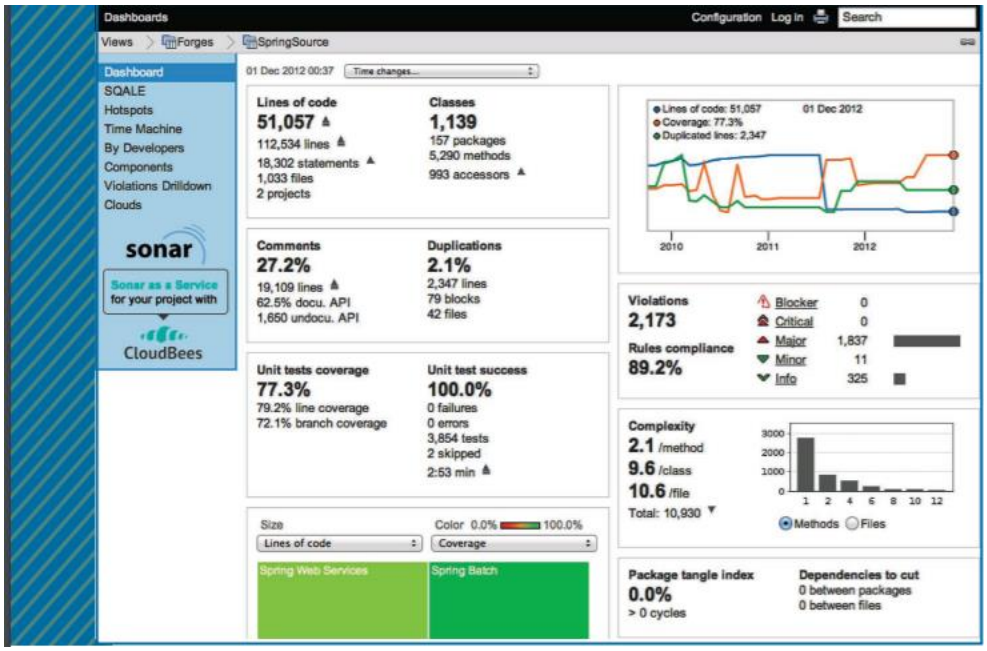
Hackystat was a huge improvement over its earlier brothers PSP and Leap. Due to this the platform was adopted into companies in the early 2000s with some welcome success. On the other hand however a few problems were found Hackystat such as its ethical side on data. As time went by employees were not happy by how their data was being collected even though the system data collection was suppose to be unobtrusive. Furthermore employees also expressed concern over how their data was being used .

Project (Members)	Coverage	Complexity	Coupling	Churn	Size(LOC)	DevTime	Commit	Build	Test
DueDates-Polu (5)	63.0	1.6	6.9	835.0	3497.0	3.2	21.0	42.0	150.0
duedates-ahinahina (5)	61.0	1.5	7.9	1321.0	3252.0	25.2	59.0	194.0	274.0
duedates-akala (5)	97.0	1.4	8.2	48.0	4616.0	1.9	6.0	5.0	40.0
duedates-omaomao (5)	64.0	1.2	6.2	1566.0	5597.0	22.3	59.0	230.0	507.0
duedates-ulaula (4)	90.0	1.5	7.8	1071.0	5416.0	18.5	47.0	116.0	475.0

Hackystat in use

OVERALL

Over the past few years, services for software product analytics have become popular, with offerings from DevCreek, Ohloh, Atlassian, CAST, Parasoft, McCabe, Coverity, Sonar, and others. These services' analytics are typically built from one or more of three basic sources: a configuration management system, a build system, and a defect-tracking system. The image below shows a display from Sonar for the SpringSource project, which is representative of this type of service.



Overall all these platforms try to keep overheads low and data collection automatic so as to make the most out of the time of the developer while providing them with an easy to use interface.

Algorithmic Approaches

MACHINE LEARNING ALGORITHMS

One way of going about measuring the software engineering process using an algorithmic approach would be to use machine learning algorithms. These algorithms would essentially be a way of automatically looking at data and then analyzing. This would then help the "machine" learn to identify any problems within projects and any patterns popping up and thus help to fix them.

In the paper APPLYING MACHINE LEARNING ALGORITHMS IN SOFTWARE DEVELOPMENT by Dr Du Zhang we see a few ways in which machine learning can be used in software engineering. A few uses that stick out to us are requirements engineering, validation and software defect prediction. Validation in particular is quite interesting as it is basically a way of using machine learning algorithms to make sure that the implemented software system conforms to its software requirements. This could help with identifying the metric of quality in the system that has been built and making sure that the product conforms to company quality standards. Furthermore software defect prediction could also be used to predict and some of the errors that developers would make in the system. This would help greatly with productivity as it could help avoid them and teach people not to make them, quality as there would be less errors in the end and also with deadline requirements as less time is spent debugging.Overall machine learning could be used to measure the errors that humans would generally generate when writing up software and thus help prevent them and improve quality as well as productivity.

ETHICS

When it comes to measuring the software engineering process ethics seem to be sometimes inside of a grey area. Data nowadays is very important to firms and to individuals. And besides being very important, data also needs to be handled with

much care and responsibility especially with the likes of the GDPR coming in where simply leaving a piece of paper lying around with someone's details on it could land a company with a big fine. Furthermore the more confidential or private the data the harder it is to get as people are uncomfortable giving it away and want to know what its being used for.

This brings the dilemma from the paper Searching under the Streetlight for Useful Software Analytics by Philip M. Johnson of data usefulness. Generally the easier to get the data the more generic it is and is of less use. On the other hand harder to get data provides more in depth original knowledge and is of much more use. However in here comes the question of where is it still ethical to gather this data and use it. At one point is the data too sensitive to use or will employees draw a line. It is an important question of measuring the software process and I believe that as time moves on and we get to the next generations this will be less of an issue as I find on a daily basis that the younger generations have less and less problems providing their data for research, business etc.

Conclusion

Overall I feel like there are many different ways to measure the software engineering process with plenty of measurable data to do so. Platforms are plentiful and we can even get machines to point out patterns to us. This is important as companies will be looking more and more to gather this type of data so they can improve their software developing. However the main enemy of this process at the moment would be ethics and this is not just in software engineering but also in many other businesses such as sales. The only way to get past this is to ensure data is safe and handled responsibly by the platforms in place.

REFERENCES

Anon, (2018). [online] Available at: https://www.researchgate.net/profile/Jan_Sauermann2/publication/285356496_Network_Effects_on_Worker_Productivity/links/565d91c508ae4988a7bc7397.pdf [Accessed Nov. 2018].

Citeulike.org. (2018). *Searching under the streetlight for useful software analytics*. [online] Available at: http://www.citeulike.org/group/3370/article/12458067 [Accessed Nov. 2018].

En.wikiversity.org. (2018). *Software metrics and measurement - Wikiversity*. [online] Available at: https://en.wikiversity.org/wiki/Software_metrics_and_measurement [Accessed Nov. 2018].

Johnson, P. (2013). Searching under the Streetlight for Useful Software Analytics. *IEEE Software*, 30(4), pp.57-63.

Nextlearning.nl. (2018). [online] Available at: http://www.nextlearning.nl/wp-content/uploads/sites/11/2015/02/McKinsey-on-Impact-social-technologies.pdf [Accessed Nov. 2018].

Pdfs.semanticscholar.org. (2018). [online] Available at: https://pdfs.semanticscholar.org/6fe8/4ae5a785391e1f8805eb261cc7cb1bf8a2fc.pdf [Accessed Nov. 2018].

Pdfs.semanticscholar.org. (2018). [online] Available at: https://pdfs.semanticscholar.org/6fe8/4ae5a785391e1f8805eb261cc7cb1bf8a2fc.pdf [Accessed Nov. 2018].

YouTube. (2018). *The future of the professions: how technology will transform the work of human experts*. [online] Available at: https://www.youtube.com/watch?v=Dp5_1QPLps0.