Software engineering measurement report

By Eric Looby

Student ID: 17327409

Introduction

The world of software engineering has many elements which can be measured and assessed to check the efficiency of software developers on certain tasks. Employers need to be able to check the level of output their developers are putting out and the impact it has on the speed and quality of code that they create. For this report I will be studying this topic under four different headings: measurable data that can be found in the software engineering process, computational platforms that are available to measure data, the algorithmic approaches that are available to measure software development and the ethical concerns surrounding the analytics involved in measuring software development.

Measurable data

There are many forms of data which can be created from the work of software engineering. This data could include the amount of code that engineers submit, the amount of changes they make, the number of commits they make, the amount of working code they contribute or any other possible number of pieces of data that could be created in this process.

A basic metric that can be measured such as the amount of lines of code that have been by a developer can be dependent on a plethora of factors. A developer may produce a large quantity of code but if this code is not bug free, meaningful or tested it will not fit in to the project and help to solve the problems that are trying to be solved at hand. Therefore, it would be beneficial if a developer output could be put through the necessary unit tests and code coverage to ensure that the code is enough and that it does not contain any fatal errors that could lead to troublesome bugs further down the development cycle.

The impact of a developer to a software project can vary based on the size and relevance of their contribution. If a developer has contributed dozens of lines of code which could have been eradicated by adding a simple function, this could be more productive because they have committed more lines of code instead of implementing a function. It would be more productive if the developer has considered dependencies which may exist between the files in a project before adding more code to a project and only add code when they are sure it is necessary.

As there is no one singular metric which can be used to determine the quality of a software project then we must use one of two types of software metrics: indirect metrics or direct metrics. The IEEE standard 1062 states that a direct measurement is “a metric which does

not depend upon a measure of any other attribute" (IEEE, 1998). An example of this is number of commits to a repository. Before a commit no other information is needed for it to occur. The creation of a metric takes in one value and outputs another value. However, for an indirect metric it relies on other values to create it. An example of an indirect metric is the bugs which may be present per line of code. Direct metrics are more accessible to collect, but indirect metrics usually give employers more useful data to be examined. If a lot of time is spent gathering metrics this could increase the overheads for the project at hand. If a direct measurement can be assumed to be valid and we have any indirect metrics in terms of these direct metrics, then we can assume that they are valid as well.

The IEEE standard has given the mean time to failure as an example that can be used for a direct metric. There is a paper by Kaner and Bond (Kaner & Bond, 2004) on this topic. In this paper it looks at a scenario where there are two different users. One of these users uses the software in question for a 24/7 period, while the other only uses the software for ten minutes every second day. Let's imagine that the mean time to failure for each case is 2 weeks. The performance of the first user is quite reasonable, but the performance of the other user is abysmal when we compare the two sets of results. One point that this paper proves is that it is hard to get direct metrics that are truly valid. Computers on their own are simple, but " as soon as we include humans in the context of anything that we measure – and most software is designed by, constructed by, tested by, managed by, and/or used by humans – a wide array of system-affecting variables come with them. We ignore those variables at our peril" (Kaner & Bond, 2004). Due to so many factors it is usually easier to make measurements when we decide to ignore certain factors, even if this is inaccurate.

It can be difficult to collect data without having negative implications on the software engineering process. Philip Johnson has written in his paper 'Searching under the Streetlight for Useful Software Analytics' that:

"after many years of exploring different approaches to analytics, we conclude that the field isn't converging on a single best approach, nor are the latest approaches intrinsically better than the earlier ones. Rather, the community has been exploring the space trade-offs among the expressiveness, simplicity, and social acceptability" (Johnson, 2013)

This summary of his work is that there is no current solution for collecting data that we want to measure that is perfect. We will have to face trade-offs, depending on what data we hope to accumulate.

One thing which may not be considered in terms of measurement is testing. Test Driven Development can add "an additional 15-35% increase in the initial development time (George & Williams) but also "78% of developers thought that test driven development improves the overall productivity of the programmer" (George & Williams). Even though test driven development can be a great way of measuring a developers' productivity in terms of usefulness, lets imagine a situation where the code is made up of tens of thousands of lines with a complexity reaching the millions. It may be possible to refactor this

code into a simpler structure, but the time it will take will be to detrimental to the progress of the rest of the project.

Developers need to be aware that with the right perception any data they produce can be measured even how the data was created or the purpose of its creation. One issue that could arise is if there is an over abundance of information collected which may not be exactly relevant such as the exact time between commits or the time they spend on break.

Computational platforms available to perform this work

There are many computational platforms that have been developed in the last few decades to help employers to measure their software developers. One of the main tasks these platforms need to get right is the balance between getting relevant data and ignoring irrelevant data.

One platform that does this is Hackystat. This platform involves software and sensors which are connected to development tools that can continuously monitor and send data which is sent to a server where managers can access this data and partake in detailed data analysis. The university of Hawaii conducted a study that showed "developers repeatedly informed us that they weren't comfortable with management access to such data, despite management promises to use it appropriately" (Johnson, 2013). The main consensus was that the developers would prefer knowledge beforehand of who has access to their work habits, rather than having it circulated to different managers without their prior consent. The data Hackystat accumulates can be very detailed but it only covers business metrics, there are other computational platforms which have gone beneath the surface and hope to look at the relationship between an employees' health and their productivity while in work.

Hitachi, a Japanese company, have claimed to invent a sensor which can use employees' movement levels to measure their levels of 'happiness'. It monitors all movements, even while sitting, the time they spend walking and even how long they spend conversing with others. Bank of America have put a similar product on trial in there call centres, called humanyze which is said to take 'sociometric analysis'. The results from this analysis showed that workers should have more group break times. Changing this aspect of the call centre improved hold time by 23% (Humanyze, 2017). Without using this platform, they would not have improved productivity by over 20 percent.

Stress management company BioBeats produce a wristband which monitors heart rate and stress levels and then computes the optimal stress level for productivity. It can even check if an employee is overly stressed with their workload or if they can take on more work. This product could allow companies to look after developers who may be pre-dispositioned to certain cardiac conditions which could even become fatal if the employee is put under stress. Without proper and unbiased use this product could be an interesting addition to software engineering.

Although its main function is to serve as a repository for code. Github is bountiful with metrics and analysis into the software development cycle of projects and the team who are working on it. A commit alone on Github can measure a lot about a developer. Github commits can be used to give a succinct view of the output of any given developer on a team and to gather data such as any lines of code they have added or deleted from a project. Commit statistics that we can see do not give enough detail to let us see what contribution the developer has had on the overall project they are working on. One major insight commit data can provide is the importance of a developer to a company due to their contributions to the code base.

Automated tools that can plug in straight to a developer's IDE and the repositories they use to house their projects code can also be quite useful for analysing any metrics within the organisation. An example of a platform like this TestRail, a product which can be integrated directly into both Github and Jira. TestRail will not retrieve source code productivity metrics such as individual productivity regarding any lines of code they have committed to a project. Instead it enables teams to monitor there testing metrics, allowing both developers and project managers to monitor the quality of their testing, code coverage, test results and other metrics which are related to the individual tasks that they are currently working on. By using the information that TestRail provides, both developers and project managers can reduce the quantity of bugs that they produce in a project, thus increasing productivity overall by reducing the need to spend time fixing bugs that may arise down the line. (Test Case management – TestRail, n.d.)

Tools such as Jira by Atlassian are used through-out the software engineering industry today. Jira allows managers to map out and design 'sprints' into tasks and then assign these tasks to individual developers within their team using a scrum board. These tasks can be tracked throughout the entire lifecycle of a single sprint as the developers are updated on the progress of each individual task as they work their way through the required tasks. This gives developers on a team visibility of what will be expected of them next to allow them to deliver maximum output in the shortest amount of time. Jira also provide in-house reports for the workflow related to each sprint. Product managers can gain access to these reports and view charts and metrics such as burndown charts, which show the total work remaining and the likelihood of achieving the goals they have set out in the sprint. Jira can also be integrated with Slack and Github, to allow developers to get their code reviewed faster and reduce any overhead time spent on getting the code from development to production. (Jira Software – Features, n.d.)

The great thing about having so many different computational platforms operating is that each company will try to find something which helps them to stand out from the competition. Whether this is providing an assurance to the privacy of the data that they help to collect or even the way they process the data, the competition in this market will create stronger tools for 3rd party software developers to be able to get the benefits from these companies. Although each platform has their own advantages and disadvantages it is going to be up to the engineering leads in the company to decide what data they really need to be measuring and thus select the correct platform(s).

Algorithmic approaches available

There are many algorithms which can aid humans in measuring software engineering. Due to the nature of an algorithm, you do not have to worry about any bias from mathematics that a human measuring data may have which inhibits them from looking at the measurement in a purely objective fashion.

One measure which used to be in use was the equation:

productivity = ESLOC/PM

Where,

ESLOC is equal to effective source lines of code

PM is equal to person month.

ESLOC must be equal to or greater than the number of source code lines that have been changed or created. This metric is at its most effective when all the source code to be measured is new. In the real world, most software projects are made up of code which has been updated over months or even years, with many different developers being involved along the way. This is mainly due to bugs or updating systems which needs to be done on a continual basis. To make the ESLOC equation more inclusive it has seen revisions to consider the continuous nature of software engineering projects. For a developer to make modifications to any existing code, they must first learn what the underlying architecture and systems are behind the code. Then they must understand what the code is meant to do. If the code does not have enough documentation by its past contributors, the current developer may need to do some reverse engineering to learn about how the code works and only then can they make the modifications they need to make. This code may need to be recoded into a different programming language or be compiled with a new compiler.

An adaptation adjustment factor is applied to modified code to consider complexities which may arise using the following formula:

Adaption Adjustment Factor = 0.4Fdes + 0.3Fimp + 0.3Ftemp

Where,

Fdes = percentage of software that is required to be redesigned or reverse engineered

Fimp = percentage of software that must be re-coded

Ftemp = percentage of software that requires regression testing

Using the adaptation adjustment factor the ESLOC variable can be re-written in the following way:

ESLOC = = Snew + Smod +Sreused * AAF

Where,

Snew = new lines of code

Smod = modified lines of code

Sreused = reused lines of code

AAF = adaptation adjustment factor

Although in theory the equation above may seem like it could be useful, it is not as accessible to count the ESLOC for developers in an organisation. External scripts/tools need to be utilised to retrieve statistics from the organisation's code repository and then this information had to be used to calculate the ESLOC metrics. When ESLOC metrics are used over multiple projects within a single organisation they can be effective in allowing a cross-comparison between the productivity of developers throughout a given development task. (Chatterjee)

The Halstead complexity measures were introduced by Maurice Howard Halstead as part of his treatise on establishing an empirical science out of software engineering. This gives us another algorithmic approach which can be used to measure software engineering.

Considering any given software problem. Let:

n1 = number of distinct operators (e.g. +, = …)

n2 = number of distinct operands (e.g. a, b)

N1 = total number of operators

N2 = total number of operands

With these numbers we can derive several measures that can combine to analyse problems from a mathematical perspective.

Using the numbers above we will be able to define the following:

Program vocabulary: n+n1 + n2

Program length: N = N1 + N2

Calculated program length: N' = n1 $\log_2 n1$ + n2 $\log_2 n2$

Volume: V = N x $\log_2 n$

Difficulty: D = $\frac{n1}{2}$ x $\frac{N2}{n2}$

Effort: E – D x V

The difficulty formula relates to the difficulty developers could have in writing and understanding the program which would involve conducting first a code review and then writing supplementary code after. The effort formula is meant to be actual coding time using the following relation:

Time required to program: $T = \frac{E}{18}$ seconds

It is also possible to estimate the time that is required to test the following equation:

Time required to test: $U = \frac{E}{K}$

Note: k is the Stroud number, which has an arbitrary default of 18. It can be changed to fit a teams' own test conditions based on how critical the code needs to be.

Halstead also derived a metric for delivered bugs which tried at estimating the number of bugs that may be found in a developer's proposed solution.

Number of bugs: $B = \frac{E^{\frac{2}{3}}}{3000}$

While it was revolutionary at the time Halstead first debuted his findings and can be somewhat useful in a 21st century software engineering project, Halstead's metrics do not have as much practicality due to modern software development tools and programming languages which can aid developers in writing their programs much faster and with less errors overall. They are still quite useful to be used as a benchmark for analysing actual metrics which can appear in software engineering projects such as bugs produced and time taken but should be considered with some hesitation and tested over multiple projects before approaching a conclusion upon their relevance to any particular developer or team (IBM Knowledge Center, n.d.)

The main benefit of implementing algorithms to help measure software development is that you can get passed the limitations which are in place based on human observation. If a manager is to decide how productive a developer is performing on a given project, they may bring personal bias into their analysis but when algorithms are used there is no arguing with the results that they produce. Although some of the algorithms I have discussed above may seem archaic in parts they can still serve a purpose if they are being used to help find

Ethical concerns surrounding this kind of analytics

Ethics is not the first thing you think of when you hear the word engineering. Engineering is a cold, precise method of solving problems and producing output. However, we need to have an appreciation that behind every feat of engineering there is a real person behind it. Employers need to keep in mind that the developers that they have employed are people

and not machines. They should not expect their employees to be as easy to assess as machinery. Also, unlike a computer, humans would prefer some sense of privacy regarding their workplace.

A big driver for the pursuit of these types of analytics is profit. some employers may argue that retrieving more data about their developers can increase their work satisfaction and improve employee productivity. But do these employers really have their employee's best interests at heart. The invasive nature of these measurements can impinge on the privacy of their workers.

Systems and methodologies that have been put in place just to keep developers under scrutiny as they are measured from the time, they walk into work could lead to a lot of paranoia in the workplace. This could affect the mental health of these developers and further increase their level of stress at work, on top of everything else they must deal with. It would seem counter intuitive for employers to induce such an anxious environment into their office. This anxiety could lead developers to commit any code just to keep up with their quotas.

Another issue which arises is once an employer has started to monitor metrics of their developer's code, could this lead them to inspect their emails and their search history to check if employees are being productive or slacking off during work time. Could it even lead to them tracking other metrics offline such as the amount of time their developers spend away from their computers, how long they take their breaks for and who they are conversing with. If the workers feel they are being scrutinised to closely they may feel like they are being monitored like they are in a dictatorship.

As I have mentioned with the survey about the developers being scrutinised under Hackystat, these employees felt a huge sense of uneasiness with the level of data collection taking place. I would consider this to be less of an ethical issue because the only metrics that are being measured are related to business relevant data on the step's employees take in completing their work. When data collection is increased to include personal data such as the health of an employee, such as with Hitachi, Humanyze or Biobeats, this may be infringing more strongly on the privacy of developers working alongside these computational platforms. A senior policy analyst with the American Civil Liberties Union, Jay Stanley, had stated: "employers have a legitimate right to monitor their workers' performance, but that right doesn't extend outside the workplace, and it doesn't extend inside people's bodies to physiological measurements. If competition begins pushing companies to accept such intrusions, that's when it's time for regulatory protection" (Heath, 2016). Heath makes it very apparent that some companies may feel themselves that it is unethical to partake in such invasive measuring of their developers but that it may be necessary to stay competitive in the market against rival companies.

There is also the train of thought that if employees could choose whether or not to participate in allowing their employers to be able to measure their work activity these freedoms would also be ignored if the benefits to employers for being able to closely monitor their workers was too great. Dr.Chris Bauer from the University of London has

made an observation that, “similar to sports science, it’ll eventually lead to a situation where those that have invested and made strategic decisions to integrate these kinds of technologies into their workforce will be more competitive” (Heath, 2016). This could lead to a reality where developers will either must sacrifice their freedoms to work for larger companies, be a freelance worker or else be unemployed.

Some of the bigger software companies like Facebook and Google have decided to try and be more lenient in their approach to the measurement of their employees. They believe by having a more comfortable work environment, this will encourage workers to perform at a higher rate as opposed to strictly enforcing workers to produce products at the highest rate. If workers feel like they have more control and are not being analysed as closely they will feel like their privacy is protected and that they are being treated like human beings not machines. This will encourage them to work harder to keep these freedoms.

It is interesting phenomenon that websites have become stricter with the data they can collect on customers ,most notably seen with the rise of GDPR compliance for websites operating in the European Union, but it seems they do not have any issue monitoring the data of their own workers. It is most apparent that the ethical concerns surrounding the analyse of developers on a software project is a hugely topical problem now which will get worse before there is some form of legislation put in place to put a common procedure in place for what data is explicitly allowed to be monitored in the a software company.

Conclusion

Although there will always be people with gripes about the increased surveillance of software it is a necessary step to allow software engineering to be measured more effectively. Project leads and management need to be able to get an idea on when they can get certain projects completed based on the data that they can measure about their developers and the projects that they are managing. The amount of data that is measurable is immense. Whether they choose to monitor the employees through their commits and lines of code alone or employ one or many of the computational platforms available to help them to tailor the type of data they wish to measure, they should keep the best interests of their developers at the core of there analyse. With necessary and relevant measurements, the software engineering process can be streamlined and improved to increase productivity for both management and developers.

References

Biobeats (2017) help keep employees healthy and productive. Retrieved from biobeats.com:

https://biobeats.com/corportate-wellness/

Chatterjee, D (n.d.). Report – Measuring Software Team Productivity. Retrieved from Berkeley.edu: http://scet.berkeley.edu/

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

Heath, N. (2016, April 11) Every step you take: How wearables will help the boss keep tabs on staff. Retrieved from:

https:// www.techrepublic.com/article/every-step-you-take-how-wearables-will-help-the-boss-keep-tabs-on-staff/

Humanyze (2017). U.S. Bank Improves Call Center Productivity. Retrieved from humanyze.com:

https://www.humanyze.com/cases/major-us-bank/

IBM Knwoledge Center. (n.d.). Retrieved from: https://www.ibm.com/support/knowledgecenter/SSSHUF_8.0.0/com.ibm.rational.testrt.studio.doc/topics/csmhalstead.htm

IEEE. (1998) IEEE std. 1061-1998, standard for a software quality metrics methodology, revision. New Jersey: IEEE Standards Department

Jira Software – Features. (n.d.) Retrieved from Atlassian.com:

https://www.atlassian.com/software/jira/features

Johnson, P.M. (2013, July) Searching under the streetlight for useful software analytics. IEEE software. Retrieved from:

http://csdl.icsi.hawaii.edu/techreports/2012/12-11/12-11.pdf

Test Case Management – TestRail. (n.d.). Retrieved from:

https://www.gurock.com/testrail