CS3012: SOFTWARE ENGINEERING

PROF. STEPHEN BARRETT

"Deliver a report that considers the ways in which the software engineering process can be measured and assessed in terms of measurable data, an overview of the computational platforms available to perform this work, the algorithmic approaches available, and the ethics concerns surrounding this kind of analytics."

Student Name:	Benjamin Colwell
Student Number:	15325781

03rd December 2017

How long is a piece of string? There is a tremendous value in being able to measure things, it's human nature to want to quantify, compare and logicise, however it isn't always possible. In the case of software engineering the benefits of being able to tangibly measure performance is most prominent - highlighting problem areas before they become problems, objectively comparing productivity of individuals so that you might reward and reprimand accordingly, accurately showing the status of a project so that we can better predict schedules and deadlines. The benefits are countless, but is it even possible and, somewhat more importantly, should we even be trying?

There are many who would say that attempting to measure such a complex system is not only difficult, but impossible. Esteemed British software engineer, Martin Fowler, once said, "I can see why measuring productivity is so seductive. If we could do it, we could assess software much more easily and objectively than we can now. But false measures only make things worse. This is somewhere I think we have to admit to our ignorance". (Fowler, 2003) Not only this, but there is also the argument that the measuring process itself has a significant negative impact on performance. But, lets assume for a moment that it is an achievable task, what exactly would we measure?

Ways the software engineering process can be measured.

The software engineering process (SEP) is a complex process, attempting to measure it is completely different from company to company, project to project and even from individual to individual. It needs to be able to adapt to suit each situation. There is no silver bullet for measuring SEPs.

The most common way to attempt to measure SEPs is using software metrics. In essence, these are just variables of data we gather. Due to how complex a SEP can be there is no one metric we can use to measure productivity or performance of a software

engineer. It is a very difficult thing to quantify and so instead we try to get the best understanding we can by combining the most relevant metrics for the project. There are two types of software metrics, direct metrics, and indirect or derived metrics.

A direct measurement, as stated by the IEEE standard 1062, is "a metric that does not depend upon a measure of any other attribute." (IEEE, 1998) In other words, it is a metric that does not need other pieces of data to create it. For example, lines of code, or number of commits to a Git repository. Neither of these pieces of data need to know anything else to be 'calculated' or created. We can say that its domain is singular. The creation of this metric takes in one value and outputs one value, it is one to one. In contrast, a derived metric is one who's domain is multiple, it relies on other values to create it. For example, defect density (bugs/lines of code) or programmer productivity (lines of code/programming time). In general, direct metrics are much easier to collect than derived ones, but usually give us less useful information than derived metrics and so there is a trade-off. More time spent gathering the data results in higher overheads for a project and the gathering process itself can prove negative to the SEP.

A direct measurement is assumed to be valid, and thus, if we define derived metrics in terms of direct metrics we can also assume they are valid. (i.e. If we assume the direct metrics of number of bugs and lines of code to be valid, we can thus derive that defect density (bugs/lines of code) to be valid.) However, it is often not this simple. Direct metrics are often much more complicated than they first seem.

Due to the varied nature of how humans will interact with software, it is incredibly rare for a direct metric to be truly accurate. The IEEE Standard gives mean time to failure (MTTF) as an example of a direct metric. It's just a measure of time so it must be direct right? If we look a bit more in depth we will see it is a bit more complicated. Let's take an example from a Kaner & Bond paper on the topic (Kaner & P.Bond, 2004), it explores the scenario of two different users, one of which uses the software in question 24/7, while the

other uses it for 10 minutes every second day. Now let's say that the MTTF is two weeks in both cases. The performance of the first user is not too bad, but the performance for the second user is horrendous in comparison. This is an exaggerated example, but it does shine a light on a major problem with software metrics in general. It is incredibly difficult to get truly direct metrics that are entirely valid. When working exclusively with machines it is much simpler, 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 & P.Bond, 2004) This has been a widespread problem for scientists and mathematicians, attempting to quantify and logicise the world we live in simply has too many contributing factors and the only way to advance is to simply ignore some of them, however this does introduce error. Despite this, software metrics are still the most widely used tool for measuring the SEP and as such, the process of how we collect these metrics is one that has been widely explored.

As previously mentioned, it is a difficult task to accurately gather useful data without negatively affecting the SEP. Things like additional overheads, invasion of privacy and even a less comfortable working environment are all different drawbacks of a few gathering methods. In the paper, "Searching under the Streetlight for Useful Software Analytics", Philip M. Johnson writes 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 earlier ones. Rather, the community has been exploring the space of trade-offs among expressiveness, simplicity, and social acceptability." (Johnson, 2013) I think this sums up all that we currently know about the best ways to collect the data we are after. It's all about trade-offs, there is no perfect solution (or at least we haven't found one yet).

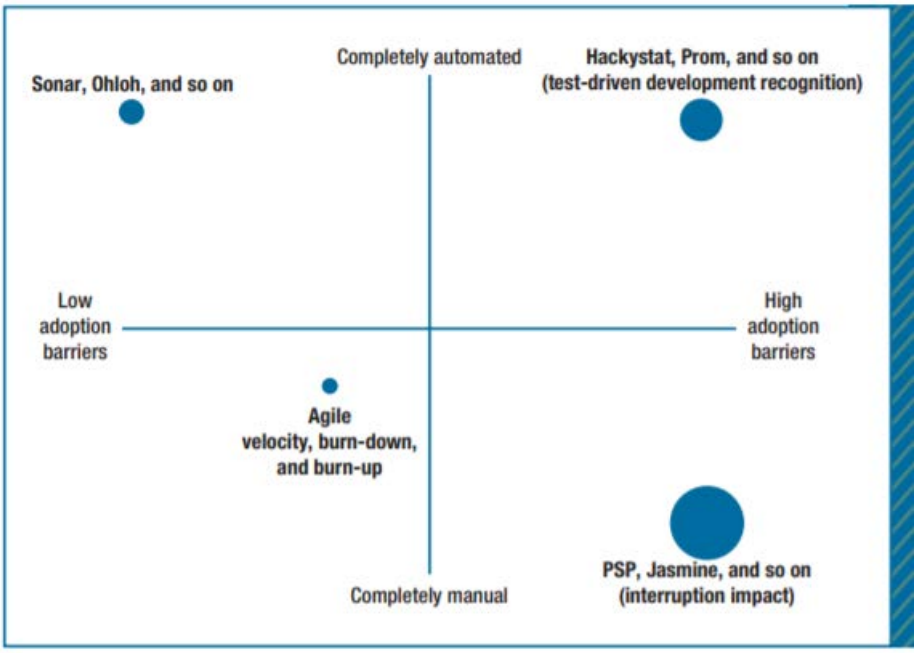

Fig.1 A classification for software analytics approaches

(Johnson, 2013)

If we look at this graph from the same paper, we can see this concept visualised for our better understanding. The X-axis plots the adoption barrier, or in other words, how difficult it is to get the system setup and running. The Y-axis plots how manual the process is which in turn contributes to how heavy the overheads incurred are. Finally, the "level of generality" or the extent of useful data gathered, is visualised by the size of the circles. Plotted on the chart we can see some of the most popular software analytic approaches. PSP or Jasmine, for example, would appear to be the best as it gathers the most useful and in-depth data. However, it is also extremely manual with very high adoption barriers. In PSP, the software developers must manually fill out numerous forms such as target dates, project plans and defect logs and because of this the barrier to entry is extremely high, few to no developers are going to want to spend large amounts of their time filling out forms on how a project is going when they could be doing the project. Contrastingly, at first glance Sonar and Ohloh might seem to be among the worst approaches as they gather far less meaningful and useful data. However, unlike PSP they are almost entirely automatic and have a very low barrier to entry. Developers can spend their time developing and managers still get a decent amount of useful metrics. As a

result, it is used by far more companies than PSP is. While all these approaches have pros and cons, the data gathered is useless or even worse, incorrect, if we don't fully understand and analyse it correctly.

Analysis of Data

Analysis of software metrics is key to making use of the data being gathered, it's a useless number to know how many lines of code a developer has written if we have no gauge on what an average result for that would be. To derive meaning from our data some form of analysis must take place. Many computational platforms for software analytics will do some basic analysis for you. Take the popular Sonar for example, alongside gathering of the data it also presents us with a dashboard to help the user derive some meaning from their data. The graphs and statistics created from this data give meaning to it and make it useful, whereas before knowing that Joe Bloggs had written 200 lines of code this week was useless, we can now see that all of Joe's co-workers have written 1000+ lines this week, and so perhaps we need to act on this or explore it further.

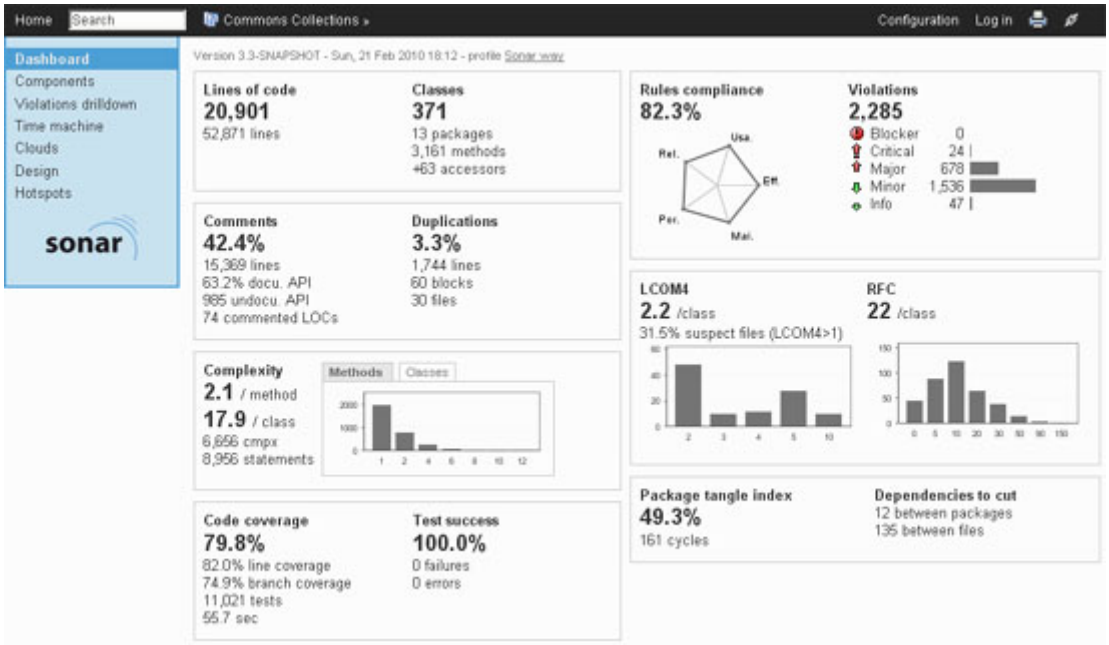

Fig. 2 Sonar dashboard example

(Gaudin, 2010)

While this is useful, it is extremely limited. "Managing source quality looks like the natural next step for development teams in their effort of industrialization. Sonar enables to reach this objective with few efforts and with fun." (Gaudin, 2010) Its goal is to simplify, and it does so well, but in many cases, it won't be detailed enough. As we discussed before, the SEP is extremely varied, and so in many cases, analysis that is more personalised to the project is required.

One of the most fundamental things when attempting to analyse software metrics is to understand what metrics this project most values. The importance placed on different pieces of data will have an extremely high variance from project to project. A customer database for a bank will value code security and reliability higher than anything else, while some new app idea rushing to market might value programmer speed most, the weight given to each metric all depends on the project at hand. A common way to make use of our metrics is the 'Goal, Question, Metrics' approach. This entails first defining a goal we are trying to achieve, translating that into the questions we want the answers to, and then finally analysing and manipulating our data to produce the results we need. The following example is found in the goal question metric approach by Victor R. Basili.

Goal	Purpose Issue Object (process) Viewpoint	Improve the timeliness of change request processing from the project manager's viewpoint
Question		What is the current change request processing speed?
Metrics		Average cycle time Standard deviation % cases outside of the upper limit
Question		Is the performance of the process improving?
Metrics		$\frac{\text{Current average cycle time}}{\text{Baseline average cycle time}} * 100$ Subjective rating of manager's satisfaction

Fig 3. Example of the GQM approach

(Basili, Caldiera, & Rombach, 1999)

Here we can see that the goal is to improve the timeliness of change request processing. Translating this into some tangible questions we can act on we get, "What is the current change request processing speed?" and "Is the performance of the process improving?". Then finally, we can give an answer to these questions by performing some basic arithmetic on the relevant metrics. From some simple analysis using the goal, question, metric approach we have gone from knowing the average cycle time, which, on its own is a fairly useless piece of information, to being able to see whether our change request processing speed is improving over time – a much more useful and relevant piece of information.

To accurately produce the results to these questions it is important that we understand what exactly our metrics are measuring. Let's take lines of code (LOC) as an example. If developer X on average writes 500 lines of code a day, but developer Y only writes 400 lines a day, this must mean that developer X is a faster and more productive worker, right? Not necessarily, all we are measuring is how many lines of code they have written, and yet most often in software development if you can write something in less lines that still does the same thing it's a more efficient solution. For a long time, IBM were trying to use LOC as a method of measuring their engineers, and even how much they were paid! This, however, had one major flaw, it encouraged bad, inefficient coding. In a PBS documentary, 'Triumph of the Nerds', Microsoft exec Steve Ballmer made the point very clearly when we said, "a developer got a good idea and he can get something done in 4K-LOCs instead of 20K-LOCs, should we make less money?" (Sen, 1996) (1 K-LOC = 1000 LOC). Not only did it discourage developers from coming up with inventive and more efficient solutions to problems, but it encouraged them to lengthen their current coding practices to earn more money.

Another example of misinterpreting metrics is the scenario of the 'Mythical man-month'. A man-month is the amount of work a man get done in one month, for example let's say its equal to 10K-LOCs. In theory, if we look at this mathematically, adding a

second developer to the task should produce the same result in two weeks. Following on, if we set thirty developers to the task we should get the same result in a day. Obviously, this is ludicrous, but mathematically it is valid and there within lies the problem. The time taken for the first developer to communicate the project and divide up the work is an added external factor that hasn't been considered. It is this scenario where we get Brooke's Law, ""adding human resources to a late software project makes it later". (Frederick P. Brooks, 1996) It is important to consider this logic when using any computational platform when measuring the software engineering process.

Overview of computational platforms

We have already discussed some computational platforms, namely Sonar, however, as we mentioned, while it's very non-intrusive, it only gathers very low level of data. This goes back to the delicate balance between wanting to gather accurate high level data, while at the same time not negatively effecting the software engineering process. When we touched on computational platforms previously, the only negatively effecting consequence we looked at was additional overheads with PSP, the time taken for developers to fill out multiple forms in order to gather the data. The obvious solution to this is to automate the process within the platform so that the developers may continue working, this is exactly what 'Hackystat' sets out to do.

'Hackystat' involves a series of sensors and software connected to development tools to continuously monitor and automatically send data to a central server where detailed data analysis can be done by managers. In theory this is great, however studies done in the University of Hawaii showed that "Developers repeatedly informed us that they weren't comfortable with management access to such data, despite management promises to use it appropriately." (Johnson, 2013) They weren't comfortable with the idea of data being collected about their work habits without their knowledge, it made them feel uneasy and in turn lead to a less productive work environment. The data collected

however, while incredibly detailed and beneficial, only covers business metrics, there are many computational platforms out there that want to look further into the correlations between employees' personal habits, traits and health and their work productivity.

Why would we want to measure employee happiness? In 2012, a report in the Harvard Business Review showed that compared to people who are unhappy, it has been found that people who are happy have 37% higher work productivity and 300% higher creativity. (Achor, 2012)

A Japanese company, Hitachi, has since come out with a wearable sensor to measure employee happiness through their movement patterns. It monitors their movements, whether they're sitting or standing, how long they walk, their conversations, who they talk to, for how long, etc. Following on from this Bank of America trialled a similar product in from Humanyse, an American company also specialising in what is being called "sociometric analysis", in their call centre. The results from this test showed them that they needed to allocate more group break time as coworkers talked through any problems they were having with their work during these times. The minor schedule change resulted in an improved call centre hold time by 23%. (Humanyze, 2017)

On a different tack, stress management company BioBeats uses wearable wristbands to monitor heart rate and stress levels and compute an optimal stress level for productivity. It can then see if an employee is overly stressed with a workload, or can take on more work, and schedule accordingly.

Fig 4. BioBeats Illustration

(BioBeats, 2017)

While these computing platform tools are fascinating in theory, they have some majorly questionable ethics behind them.

Ethics

The entire concept of measuring the software engineering process is something that which the ethics of are hotly argued over. Treating and measuring employees by questionable metrics and numbers that don't tell the whole story is very dehumanising in its nature. Why is an employer gathering this data about its employees any different from a government gathering it about its citizens? Data collection is ethically, a fragile subject matter and one that needs to be treated with extreme care no matter what, but in my opinion even more so in a business setting.

It is easy to argue to the concepts of freedom of speech and privacy in the real world, but within the workplace is an area this becomes more difficult where profit is a factor. There is an argument to be made that the tangible benefits in productivity and employee health justify the questionable ethical means by which they are achieved. If an employer wants to capture data so that he can better manage his workplace to improve worker health, purely to increase productivity, is it a win-win? The company progresses and the employees are happier. I think it depends on the collection method of the data.

In the case of Hackystat, employees felt uneasy with the level of data collection, but it was entirely business relevant data on how employees completed their work and so I personally don't believe that it is as ethically questionable. However, when the data collection extends to personal data, such as in the case of Hitachi, Humanyse and BioBeats, I think it crosses the line. Jay Stanley, senior policy analyst with the American Civil Liberties Union, said: "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 pressure workers to accept such intrusions, that's when it's time for regulatory protection." (Heath, 2016) She makes a very good point about the competition between companies.

Some will argue the case that it should be optional for employees, which in theory is great, but if it genuinely shows such a significant increase in productivity, it will eventually lead to a case that businesses will have no choice but to partake if they want to remain relevant and competitive. Dr. Chris Bauer of the University of London says 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) If this is the case then it could reach a stage where employees will have no choice but to opt-in to the systems or be left jobless.

This kind of Orwellian argument of monitoring is not constricted to the software engineering world but is a global issue of the 21st century. As software engineers, it is all well and good to be continuously improving our methods and systems, but if we don't stop and think about the social and ethical ramifications that Orwellian vision is not as far away as you might think.

References

Achor, S. (2012). Positive Intelligence. *Harvard Business Review*.

Basili, V. R., Caldiera, G., & Rombach, H. D. (1999, September 29). *The Goal Question Metric Approach.* Retrieved from cs.umd.edu: www.cs.umd.edu%2F~mvz%2Fhandouts%2Fgqm.pdf&h=ATOkQ84MJWPZA5hzKBLhFsLN9_7S_Yp1-3VzZa3d2Y3a-ICv43HSrPtKTtMdiTNlDs_lug8gzb3yUG2VvlCFX_mg-rgpRVELy8aE434i4pkNSdxVMYw6nIvzXXLaIjN5JJkVPSGNz2U

BioBeats. (2017). *Help keep employees healthy and productive*. Retrieved from biobeats.com: https://biobeats.com/corporate-wellness/

Fowler, M. (2003, August 29). *Cannot Measure Productivity*. Retrieved from martinfowler.com: https://martinfowler.com/bliki/CannotMeasureProductivity.html

Frederick P. Brooks, J. (1996). *The Mythical Man-Month.* Addison-Wesley.

Gaudin, O. (2010). *Sonar - Open Source Project and Code Quality Monitoring*. Retrieved from methodsandtools.com: www.methodsandtools.com%2Ftools%2Ftools.php%3Fsonar&h=ATOkQ84MJWPZA5hzKBLhFsLN9_7S_Yp1-3VzZa3d2Y3a-ICv43HSrPtKTtMdiTNlDs_lug8gzb3yUG2VvlCFX_mg-rgpRVELy8aE434i4pkNSdxVMYw6nIvzXXLaIjN5JJkVPSGNz2U

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

IEEE. (1998). *IEEE Std. 1061-1998, Standard for a Software Quality Metrics Methodology, revision.* New Jersey: IEEE Standards Department.

Johnson, P. M. (2013, July). Searching under the streetlight for useful software analytics. *IEEE Software*. Retrieved from http://csdl.ics.hawaii.edu/techreports/2012/12-11/12-11.pdf

Kaner, C., & P.Bond, W. (2004). Software Engineering Metrics: What Do They Measure and How Do We Know? *10th International Software Metrics Symposium*. Retrieved from citeseerx.ist.psu.edu: http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=66C081877663775A062009B17114DAF8?doi=10.1.1.1.2542&rep=rep1&type=pdf

Sen, P. (Director). (1996). *Triumph of the Nerds* [Motion Picture].