Measuring Software Engineering

Software engineering is defined as the systematic application of engineering approaches to the development of software. A good software engineer will approach the development of a piece of software in the most efficient way possible. However, a big underlying question in the software engineering industry is how to distinguish between a good and a bad software engineer. In this essay I will be looking at the ways in which a manager can measure his or her employee's performance. Of course, in order to attain this information, the manager must monitor the work of each individual and use metrics to analyse this performance. This can be a difficult task as it can be hard to measure such performance. I will be looking at the different metrics that are measured and the tools used to measure these, along with what makes a productive software engineer. I will then investigate the ethics behind measuring and monitoring the work of software engineers and the code of ethics that must be abided by.

Why Measure Software Engineers?

In my opinion there are two main reasons why it is important to measure the work of a software engineer. The two reasons are, 1. Measuring the performance of an individual software engineer, and 2. Helping management to better predict the completion date of a project. I will further discuss why it is important for management to be able to use these measures.

1. **Performance of an engineer:** It goes without saying that it is important for a business to be able to measure the performance of their employees. When it comes to measuring the performance of a software engineer it gets a little trickier. In the world of software engineering and with the emergence

of agile development, the short-term goals and requirements of an engineer can change rapidly. Therefore, it can be difficult to measure the progress of a project if the requirements specification is constantly being tweaked by the client. As a result of this, the performance of an engineer cannot simply be judged by whether they have hit their deadlines. If we take for example, a good software engineer will keep close communication with their clients. If midway through a project the client is not fully happy with how the project will turn out they may pivot and alter the requirements specification. This will of course lead to the project taking longer than expected even if the engineer is working productively throughout.

2. **<u>Predicting Deadlines[1]:</u>** I understand that in my previous point I have already contradicted myself by saying deadlines of a project can often be subject to change. However, in the world of business, deadlines are still very important for firms. If we think about it logically, a business will not hire another company to write a piece of software or develop an app for them if they are told "it will be done when its done". Similarly, hitting deadlines is a good way for a company to have loyal customers as if their company is seen as reliable, firms will continue to hire them. There needs to be a way in which we can make a good estimate as to when a project will be finished. Of course, as I have already stated, these deadlines may be subject to change if the client wants to add different features and what not but it is still important for an engineer to have a goal to have the code written by.

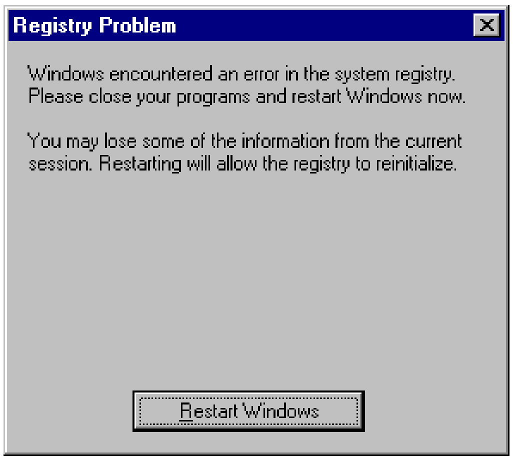
What Data Can We Measure?

There are several different metrics that we can use to measure the performance of a software engineer. The main metrics that can be used that I will be discussing are:

- Code Coverage
- Time to Failure
- Velocity
- Cumulative Flow
- Earned Business Value
- Employee Satisfaction

Code Coverage and Testing[2]: I had initially planned to have code coverage and testing as two separate headings; however, I believe that these two go hand in hand with each other. Code coverage tells the developer what percentage of the code is being tested. This helps ensure that all the code has been reviewed and that there are no undiscovered bugs in the code that may present themselves later. Testing is important as an engineer can see what is working and what is not working in their code. They can then debug their code and hopefully fix the error. Theoretically, if a program has one hundred percent code coverage and passes all the tests, there should be no bugs in the program. Code coverage and testing is an important metric for managers to use as it gives them an insight as to the quality of the code and likelihood of undiscovered bugs.

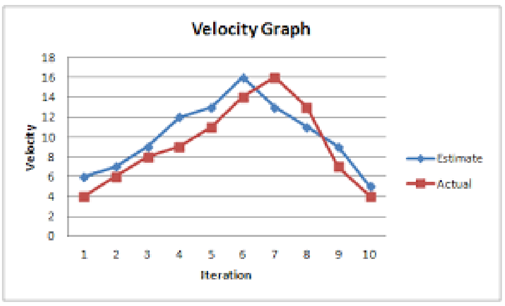
<u>Time to Failure[3]:</u> Time to Failure is a measure described by Cem Kaner in 2004. Kaner talked about the importance of this metric and dived into what this really meant. The time to failure is a measure of how often a system experiences an error. A failure does not simply have to be an error in the program that causes it to crash but can mean a number of things such as an event that wastes a user's time or one that motivates them to call support. The size of an error is also considered. A failure that causes the system to crash and forces the user to restart the system is far greater than an error that can be recovered from quickly. As you can see, the size of an error and time that is consumed by that error is important. In my opinion this is a vital metric as it gives an insight into time that is wasted by an engineer.


Registry Problem

Windows encountered an error in the system registry. Please close your programs and restart Windows now.

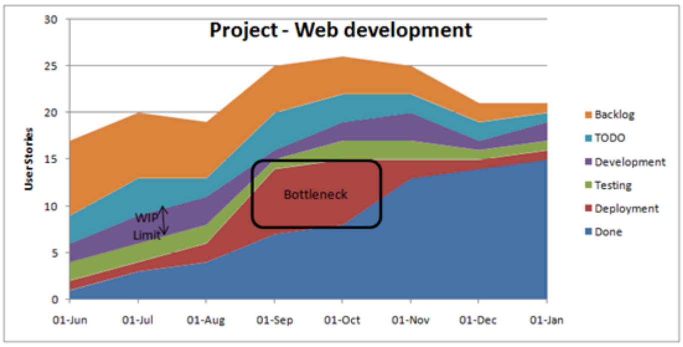
You may lose some of the information from the current session. Restarting will allow the registry to reinitialize.

<u>Velocity[4]:</u> Velocity is a key metric when measuring agile projects with a team. Velocity tells us how quickly an entire project is moving along and helps us to predict when it is finished. This is especially important when looking at projects that are being worked on by a team as it gives a look at the project overall, incorporating UI, design, etc. The faculty of computer science and information technology in UPM explains that any changes to team composition can slow down the rate at which a project is completed, and even adding a highly skilled developer can have a negative impact. This goes to show that communication between a team is vital to maximise the velocity of a project. When we look the velocity graph that I have



included we can see that it is also important to show that rate at which work is being done at each stage of the project. This can also be a useful measure as we can analyse different projects to see if work is being done slowly at a certain stage. Clearly, we can see how velocity can be a useful measure when looking at the work done by agile teams.

<u>Cumulative Flow[4]:</u> A cumulative flow diagram is an excellent tool that is used to illustrate the quantity of work being done at each stage, and what the work is being done on. A cumulative flow diagram is a useful tool for project controlling by depicting work in progress (WIP). By doing this it allows the developer to make a more accurate estimation of lead time and delivery date. As a result, we can measure more accurately how efficiently an agile team is working. Another useful tool that the cumulative flow diagram provides us with is how much of our time is taken up by a bottleneck. If this visual gives us an idea of when the project comes to a standstill, we can do a better job at getting back on track and solving an issue when a bottleneck arises.



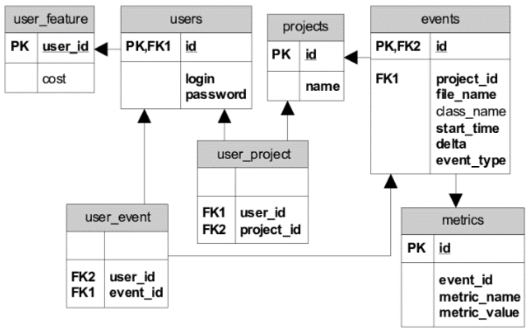
<u>Earned Business Value:</u> The earned business value (EBV) of a project is a great way for both businesses and software engineers to measure progress. To measure EBV there is a defined list of features in a project that have values assigned to them. When an agile team has successfully added each individual feature, the EBV of the project increases until all the features have been delivered and the EBV matches the price of the software that has been developed. The EBV is an important measure for a client as they do not need to have any understanding of software engineering or programming to understand how far along a project is. Again, this is just another tool used by software engineers to measure the progress of a project. This tool is more valuable to the client than the engineers.

Employee Satisfaction: As a computer science and business student, I have completed several essays where I have discussed how important high employee morale and a positive workplace is. If an employee views their workplace as a happy and healthy environment it, they will look forward to going into their workplace rather than viewing it as a chore. A study by economists at the University of Warwick found that happy employees were 12% more productive, while dissatisfied workers proved 10% less productive. If employees feel bored and undervalued, it will affect their performance[5]. This is particularly important when working in teams. In agile teams, communication is vital and without happy employees with good relationships with one another it will negatively affect the productivity of the team. According to the Harvard Business Review, "In sum, a positive workplace is more successful over time because it increases positive emotions and well-being. This, in turn, improves people's relationships with each other and amplifies their abilities and their creativity.[6]"

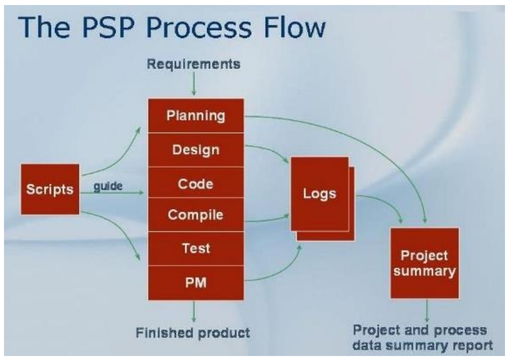
What Tools are Used to Measure Software Engineers?

Now that I have outlined what data I believe is important to measure, we need to know how we can use this data to nest understand how we can improve our processes. It would be pointless to collect this data without being able to measure it. The three tools that I will be discussing are PROM, PSP and AaaS. I will also quickly outline the best ways to measure employee satisfaction.

PROM[7]: PROM is a commonly used method for programmers to gather and review data about their work. Inside PROM the is a WebMetrics tool that performs metrics extraction. This contains tools such as language parsers that extract software metrics and are command line executable. There is then a plug-ins server that creates a file containing source code, calls a parser to analyse it and extracts results from the generated file and sends them to the PROM server that stores them in the database. The data are stored in a DBMS and can be accessed by HTML language. The database contains information such as the users, the projects that the user is working on and the tasks that they have carried out. This is a useful tool for both developers and managers. It helps developers as they can access their own data and statistics which they can use for their own personal improvement. The system only allows managers to access aggregated data at a project level. This system allows managers to keep the project under control without invading the privacy of the individual programmers. The main benefits of this tool are code and process improvements for developers and allows for easier cost management and project management for managers.



PSP[8]: The personal software process is a tool that is used by developers whereby they write out a plan of what they expect the project breakdown to look like. This plan includes metrics such as the estimated program size, the time spent on each individual phase and the defects or interruptions expected. For time measures an engineer would record things such as the time spent on any



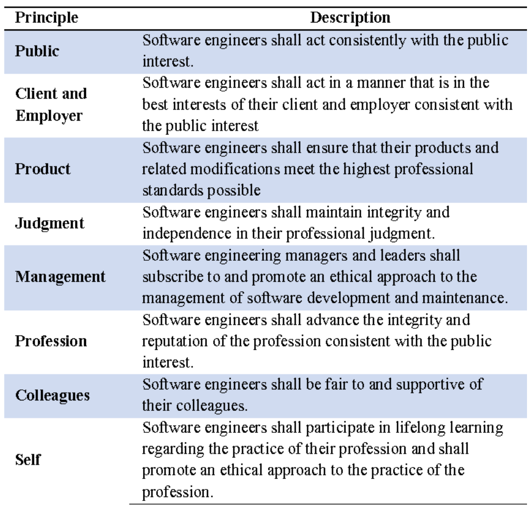
interruptions such as a phone call or a quick break taken. Recording this information allows the developer to accurately track how much time they spend working on a task. Another measure that is tracked in PSP is code quality. This is tracked by measuring several different things such as defect density or review rate. Defect density refers to the number of defects per thousand lines of code. Often a programmer will record the number of defects per, say, one hundred lines of code and use that to calculate the number per thousand. When a developer completes a project, they will then look at all the data that they have recorded and compare it to their estimates. This allows the developer to see where they are taking more time than expected and work to improve in these areas in the future.

AaaS[9]: Analytics as a service refers to the practise of using web-based technologies to carry out analysis of data. The idea of Aaas is that there are companies set up that perform this data analysis for firms. In the modern day this is very desirable as it is expensive for companies not only to analyse their data but to store the high volume of data that they are producing too. According to Forbes[10], 2.5 quintillion bites of data are produced every day. This led to the idea of cloud computing where companies could offer cloud-based solutions to the problems. This idea means that data can be stored in a cloud and can be accessed and analysed when it is needed. This idea is desirable for firms, however, if a firm were to do this it would not be very cost effective as they would have to pay for servers and handle the data themselves. Therefore, the market for AaaS has emerged as the companies that specialise in this cloud computing are able to handle a large volume of data on a large scale which is more cost effective than firms doing this individually. The major players in this industry are Microsoft, Google, AWS and many more.

Employee Satisfaction: As I have stated, I am going to give a very brief description of how employee satisfaction can be measured. The most common method of measurement is the carrying out of simple surveys. While this is a quick and easy method it may not always be accurate as it is very simple for an employee not to answer truthfully and to say that he or she is happy when they are not. A method that was discussed in our lecture was a study that was carried out recently of the device that was used to monitor an employee's behaviour and interactions with one another. While this would of course be a very accurate way of tracking employee morale, there is an important question that must be asked. This question will nicely lead me into my next topic of discussion which is, is it ethical to do this?

The Software Engineering Code of Ethics[11]

There is a code of that all software engineers must abide by. The code of ethics was adopted in 2000 by the IEEE Computer Society and the ACM who are the two leading international computing societies. After reading through the code of ethics of software engineering it is obvious to me that it has been written in order to protect both engineers and the public. The one piece that stood out most to me was 1.03. 1.03 states that a software engineer is allowed to approve software only if they have a well-founded belief that it is safe, meets specifications, passes appropriate tests, and does not diminish quality of life, diminish privacy or harm the environment. The ultimate effect of the work should be to the public good. This clearly shows that the code is written as a means to protect both the general public and other software engineers. As we know, software



Principle	Description
Public	Software engineers shall act consistently with the public interest.
Client and Employer	Software engineers shall act in a manner that is in the best interests of their client and employer consistent with the public interest
Product	Software engineers shall ensure that their products and related modifications meet the highest professional standards possible
Judgment	Software engineers shall maintain integrity and independence in their professional judgment.
Management	Software engineering managers and leaders shall subscribe to and promote an ethical approach to the management of software development and maintenance.
Profession	Software engineers shall advance the integrity and reputation of the profession consistent with the public interest.
Colleagues	Software engineers shall be fair to and supportive of their colleagues.
Self	Software engineers shall participate in lifelong learning regarding the practice of their profession and shall promote an ethical approach to the practice of the profession.

engineering has potential to do be very powerful. Of course, this can also be used in a way that may be harmful to others. Programs could be written that would gather information that would invade the privacy of people among many other harmful things. This point is again reinforced in the code of ethics in section 2.05. This section states that an engineer must Keep private any confidential information gained in their professional work, where such confidentiality is consistent with the public interest and consistent with the law. The final observation that I made while reading through the code of ethics is that there is a significant portion of it that is concerned with the improvement and development of software engineering. Particularly in principle 8, Self, there are many points made that state that a software engineer must improve their ability to understand software and their ability to produce accurate, well-written and reliable code.

My Opinion and Conclusion

To conclude this piece of work that I will be submitting I would like to give my own opinion on the matter. First, I would like to say that I do believe that it is important to be able to measure software engineers. From writing this piece and looking into what data can be measured and how it is measured, I do not believe that this data and the metrics used are particularly invasive to engineers. The metrics that I researched were code coverage and testing, time to failure, velocity, cumulative flow, EBS and employee satisfaction. The first two metrics, code coverage and testing and time to failure, are simply used to monitor the quality of code that a developer is producing. I believe that a manager should be allowed to look at this as it is an important for monitoring an employee's performance. Otherwise, how else could a manager judge the performance of an employee? And would this not lead to a bad employee being able to slack off and produce poorly written code with no consequences? Similarly. with velocity, cumulative flow and EBS, these are all just measure to view to progress of a project both overall and at individual stages. These are simple tools that managers should be allowed to use and analyse as they can significantly enhance performance by identifying where projects slow down and are not producing valuable work. Again, I do not believe that any of these measures are particularly invasive to an employee. This is also supported by how these metrics are measured. I looked at both PROM and PSP. PROM does not give all the information about a developer's work to a manager, instead it only gives general information on the performance of the engineer and progress of the project. PSP is a personal tool used by a software engineer to review his or her own work, so again, this does not invade the privacy of the engineer. AaaS is where it becomes a little more questionable. A company that is hired by a software engineering firm possesses and sorts all the data of that company. Although there is clearly an opportunity here for a large data breach, companies that provide analytics as a service are bound by contracts not to share any of this information. These AaaS companies also have very tight security in order to prevent any data breaches or hacks.

Finally, I believe that measuring employee satisfaction is ethical if the employee has agreed to the surveys that they are filling out or agree to how they are being monitored. If we look at the personal devices that were discussed in our lecture, these devices recorded the conversations and actions that employees had. In my opinion this is not ethical as it is a clear breach in a person's privacy as all their conversations are monitored. If we were all to imagine things we have said to people while having lunch with a close friend at work or in college, many of these conversations are private and should not be listened to or analysed by an employer.

In conclusion, I believe that the processes used to measure the work of software engineers is ethical. I think that we enter a grey area when we look at some of the tools that can be created by software engineers to monitor people in their daily lives. However, as I have discussed, programs that are not in the best interest of the public are not allowed to be created or used, as stated in the software engineering code of ethics.

References

1: Small Business Chronicle, Available at: https://smallbusiness.chron.com/value-meeting-deadlines-business-18475.html

2: Perraju Bendapudi, Siddharth Rana, Measuring Code Coverage, Oct 25, 2007. Available at: https://patentimages.storage.googleapis.com/9a/41/b7/ee0ac0d45edd85/US20070250815A1.pdf

3: Cem Kaner, Software Engineering Metrics: What do They Measure and How do We Know? 10th International Software Metrics Symposium 2004.

4: Taghi Javdani , Hazura Zulzalil, Abd. Azim Abd. Ghani, Abubakar Md. Sultan, On the current measurement practices in agile software development, International Journal of Computer Science Issues, 2012, Vol. 9, Issue 4, No. 3, pp. 127-133.

5: Andrew J. Oswald, Eugenio Proto, Daniel Sgroi, (2015) Happiness and Productivity Journal of Labor Economics, 33 (4). pp. 789-822.

6: Emma Seppala, Kim Cameron, Harvard Business Review (2015) [Online] https://hbr.org/2015/12/proof-that-positive-work-cultures-are-more-productive

7: Sillitti, A. Janes, G. Succi, and T. Vernazza, "Collecting, integrating and analyzing software metrics and personal software process data," in Proceedings of the 29th Euromicro Conference. IEEE, 2003, pp. 336– 342.

8: Watts S. Humphrey, The Personal Software Process, Carnegie Mellon Software Engineering Institute, 2000.

9: Dana Naous, Johannes Schwarz, Christine Legner, Analytics As A Service: Cloud Computing and the Transformation of Business

Analytics Business Models and Ecosystems, Association for Information Systems AIS Electronic Library (AISeL), 2017.

10: Bernard Marr, Forbes, How Much Data Do We Create Every Day? The Mind-Blowing Stats Everyone Should Read, 2018 https://www.forbes.com/sites/bernardmarr/2018/05/21/how-much-data-do-we-create-every-day-the-mind-blowing-stats-everyone-should-read/?sh=237c261960ba

11: Don Gotterbarn, Keith Miller, and Simon Rogerson. 1997. Software engineering code of ethics. Commun. ACM 40, 11 (November 1997), 110-118.