Software Engineering Measurement Report

CS3012
Software Engineering

Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

Jack Donal Collins
Computer Science & Business
Junior Sophister
16323107
Collinj6@tcd.ie

Introduction

This report aims to offer a wide ranging overview of the different techniques used to measure the software engineering process, the usefulness and importance of these metrics as well as an overview of the platforms and alternative algorithmic approaches that exist completed by a discussion on the ethics of analysing the software engineering process and big data in general.

Measurement

Software engineering is defined as *"the process of analysing user needs and designing, constructing, and testing end user applications that will satisfy these needs through the use of software programming languages"*[1]. Breaking down this definition we identify three core components to the activity; Analysation and Requirements Solicitation, system construction and testing. I will endeavour to measure how to measure each stage of the software development life cycle and demonstrate the intricate link between each. I shall also examine the various agile metrics used while commenting on the benefits of agile from a management and measurement standpoint. Before examining which tools and metrics should be used we must question what the key considerations are in the development of a software system and why we would want to measure the software engineering process in the first place?

John Davenport[2], of the British Computer Society notes that without proper measurement there can be in fact no measurable improvement leading to informational stagnation. Improvement schemes in software engineering, must, like all other fields adhere to a core set of principles which are needed to strategize change within an organisation. There must be a clarity of purpose in what the organisation is trying to achieve and there must be clear and easily measurable goals with trained resources and most importantly relevant reporting of these goals.

Why do we want to Measure the Software Engineering process?

Ultimately, software engineering's purpose is to complete a task to improve the process or create a new process or environment all together with the highest return on investment

[1] https://www.techopedia.com/definition/13296/software-engineering
[2] https://www.bcs.org/content/conWebDoc/7895

possible. Like in all parts of a business, it is incredibly important to measure and monitor performance with the aim of increasing productivity and reducing waste within the organisation. I.e. Is it cheaper to have 4 expensive extremely talented software engineers or is it better to have 7 less talented but less costly engineers? First we must consider the possible payoffs in achieving "perfect code";

Technical Debt

Technical Debt is the term used to describe the payoff between short term rapid development of a system rather than long term present value of a dynamic and easily modifiable system built with foresight in mind.[3] In the long term, technical debt can be cortical to a business as it may struggle to adapt/keep up with Its competitors. Tools to measure technical debt as a metric primarily focus on code quality and the overall complexity of the system.

First Mover Advantage

Robinson et Al[4] demonstrate the empirical evidence of first mover advantage in the case of Boeing. Rapid development of MVPs (Minimally Viable Products) allow small start-ups to grow rapidly without large capital investment, allowing them to test a new market or category and gain first mover advantage.

Measurement Techniques

1.Lines of Code/Source Lines of Code (LOC/SLOC)

Weiss, in 2003 noted that at the time SLOC was still a popular metric in cost estimation for the development of a system[5]. Weiss also noted that a SLOC underpinned several cost estimation models proposed by Boehm as well as it's widespread use and the comparability between projects it offered. However, he also notes several flaws with the approach, including universal agreement on what kind of lines should be counted and which should not. Measuring the number of lines of code written had long been the standard for the examination of an engineering process. Similar to asking how many bricks a bricklayer lays lines of code written tells you nothing about the overall scale of the project or more

[3] https://www.sei.cmu.edu/research-capabilities/all-work/display.cfm?customel_datapageid_4050=6520
4 https://deepblue.lib.umich.edu/bitstream/handle/2027.42/43581/11151_2005_Article_BF01024216.pdf?sequence=1&isAllowed=y
5 https://files.ifi.uzh.ch/rerg/amadeus/teaching/seminars/seminar_ws0203/Seminar_3.pdf

importantly the quality and impact of the code written. Engineers, who are trying to spoof the level of completion of a project could simply add useless lines of code that have no value add to the actual completion of the project. Having projects with higher code density can also be harmful from a Technical debt standpoint, complexity within the structure of the code can make it more difficult to add functionality later on in the development cycle.

2.Number of Commits

Measuring the performance and productivity of a project by counting the number of Commits made by developers is an arbitrary and flawed approach. The size and frequency of commits are not correlated with the completion of a project and do not offer an objective stance of the completion of a task rather it may be a better measurement of the flow and consistency of development if common principles and standards are used religiously throughout the organisation leaving no space for interpretation. In conclusion Active git commit history can demonstrate the frequency an engineer is working on a project but tells us nothing about the quality or impact or complexity of their work. More importantly commits may also remove code previously added to the repository which then tells us even less about the current state of the project in real terms.

3.Code Churn

Code churn measures the rate at which code evolves[6] i.e. It measures how a piece of code changes over time. Code churn is most useful from a project management standpoint in identifying delivery risks to the customer. Code churn, as a metric can be simplified to just a moving average of the number of lines added vs number of lines removed. One key point to note is that code churn code be positive in cases where there has been poor programming practice by another developer. Code churn in these cases could be the result of a developer finding a more efficient way to design the system which had been previously missed in the software process.

6 https://codescene.io/docs/guides/technical/code-churn.html

Code Coverage

One crucial measurement in terms testing completion for a project would be the level of code coverage completed by tests. Code coverage measures the number of lines executed in the program measured as a percentage of the entire program. The Coverage criteria includes conditional coverage, i.e. has each Boolean scenario been tested, statement coverage.ie has every statement in the code base been tested.

Observation/Keystroke tracking

One interesting way to track an individual's activities is to record every activity they carry out on their computer from words per minute to how often they use a mouse to click rather than using the keyboard, all of this information can later be analysed to determine whether correlations exists between an individuals speed and the reliability of their code determining whether engineers should be slowed down or not to improve code reliability and durability. There are often anecdotes shared in lectures about lecturers who had colleagues in workplace who were often extremely slow in the engineering process but produced code that was bug free and consistently reliable. I for one would argue that the present value of that work must be more valuable than that of a less sophisticated developer even after factoring in all of the extra time.

Self-Quantification

An increasingly prevalent trend within the industry is for engineers to define their own KPI's (Key performance indicators) to which they ascribe and what best captures their development process[7]

Cyclomatic Complexity

Cyclomatic is a quantifiable measurement of complexity of a system. The level of complexity in a program has been empirically correlated coding errors[8] which impacts on the cost of development of a project.

---

7 https://go.tasktop.com/rs/246-WDG-185/images/Understanding%20Software%20Development%20Productivity%20from%20the%20Ground%20Up.pdf

8 https://www.techrepublic.com/blog/it-security/the-danger-of-complexity-more-code-more-bugs/

$$\textit{Cyclomatic complexity} = E - N + 2 * P$$

where,

E = number of edges in the flow graph.

N = number of nodes in the flow graph.

P = number of nodes that have exit points

Logically, the theory makes sense as the more individual paths code can take through the exaction the more potential there is for unintended errors to manifest themselves. The method is popular as it is a quantifiable and definitive measure of complexity in the program. Project managers can use this measure to manage code quality. One fault with this approach is that the larger a project is the more inherently complex it is which makes it important to only compare projects on a like for like basis. For example you would expect a banks lodgement system to be more complex than a simple calculator application and comparing the complexity score for each would be frivolous.

Computational Platforms

Introduction

There are various different tools and programs that can be used to measure the software engineering process which ultimately depends on the work that has been done. For example, the process for completing a Banking Application may have a lot more legal and planning meetings than maybe the development of a calendar application. Collaboration is something that may be interesting to measure in this case, but we have found no concrete measure for.

Personal Software Process

Watts Humphry introduced the Personal software Process (PSP)[9] which provides software engineers with a disciplined approach in how they should go about completing the work in the most productive and efficient manner. PSP provides a framework for self-evaluation and best practice in software engineering.

[9] https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=5283

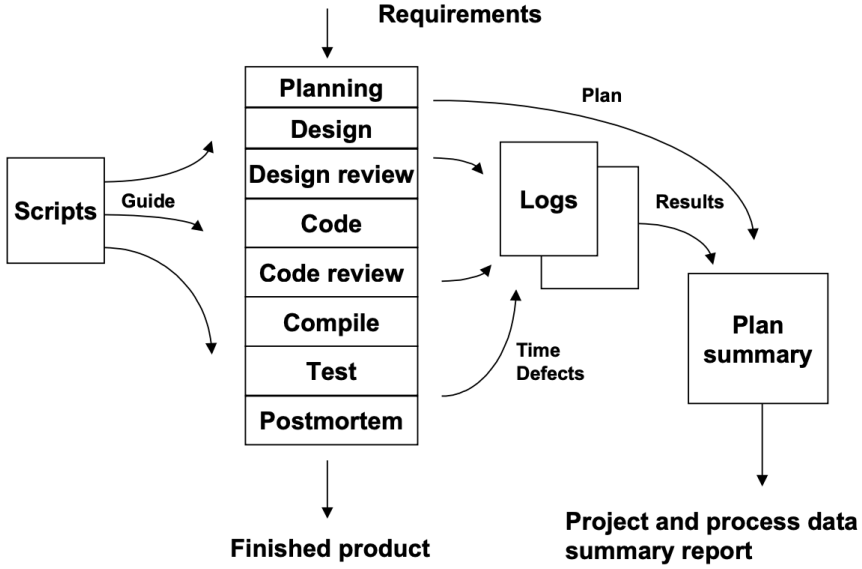

Fig.1 Process flow diagram

The PSP is similar in ways to Maslow's Hierarchy of needs, in both cases once a certain level of utility, or in this case increased performance has been achieved, engineers must then strive to improve their processes once again. The PSP roadmap can be set out by the following steps[10];

1. Define the quality goal
2. Measure product quality
3. Understand the process
4. Adjust the process
5. Use the adjusted process
6. Measure the result
7. Compare the results with the goal
8. Recycle and continue improving

It is up to the engineer to prioritise their own improvement, the PSP framework can allow engineers to discuss their own performance by measuring and monitoring their performance and adjusting the process as the identify issues.

[10] https://www.win.tue.nl/~wstomv/quotes/humphrey-psp.html

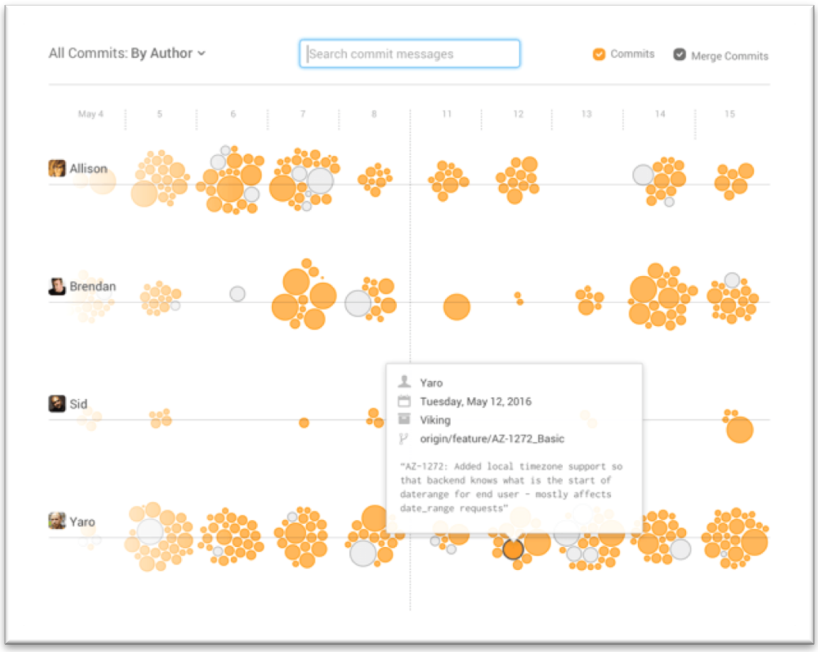

Fig5. GitPrime commit graph showing merged and unmerged gits together with code size.

GitPrime

GitPrime is a paid professional service to monitor and measure the software engineering process. GitPrime gives you commit level data on whether the commit was in fact new work, legacy refracturing an churn. These quantitate insights can help project managers estimate completion time. They can also focus on assisting/understanding the reason why some developers who have a high code churn rate.GitPrime advertise that there services can help managers determine whether agile sprint cycles had a net positive effect or whether a weekly meeting affected productivity or outcomes for a specific project. They also claim that their system can determine the riskiest commits in the scheme of the overall process which aids in recommending which commits ought to be reviewed by the entire team.

Hackystat

Hackystat is an open source framework that can be used in the analysis and evaluation of the software engineering process[11] It is designed to offer the user a visual representation or abstractions of interesting data. Hackystat has been used by both researches and the wider industry in order to help them plan, organise and control the process using an automated

[11] https://hackystat.github.io/

software suite. As most institutions use GitHub or another git derivative, they can easily integrate the repositories with Hackystat and gain valuable insight.

Humanyze

If measuring the software engineering process alone isn't enough and existential factors still exist within the process well then maybe a High-Tech start-up such has Humanyze has the answer. Humanyze offer sensors to companies to track everything their employees do throughout the day. The companies' *Sociometric* badges contain a microphone to capture speech, an accelerometer to track movement, infrared to detect to whom are you speaking with and Bluetooth to track your location[12]. Humanyze works with over 50 Fortune 500 companies across a diverse range of industry including pharma, energy and IT. The company do stress the importance of data privacy and that the system has been built to anonymise data and is designed just to be used to measure how productivity within the organisation can be improved, If you like Big brother is always watching he just doesn't know exactly who he is watching.

Algorithmic Approaches

Having established there are multiples flaws with the current measurement approaches that are available to project managers, Algorithmic approaches allow the manager to dive deeper into the data that has been collected and into the implied meaning of the data collected.

Steve McConnell, in his book diseconomies of scale in software development notes[13] that there is an exponential relationship between project complexity and effort, for example most people would assume a software engineering project 2x size would take double the work but McConnell implies that it actually takes much longer. It is no wonder then that the relationship between code size and errors is also exponential, meaning larger projects require a large amount of Quality control. The lack of information we have pertaining to the software engineering process in our current simplistic models forces us turn to statistical models for answers.

[12] https://economictimes.indiatimes.com/tech/hardware/protecting-individual-privacy-is-important-ben-waber-cofounder-humanyze/articleshow/64045690.cms
[13] https://blog.codinghorror.com/diseconomies-of-scale-and-lines-of-code/

Bayesian Belief Nets

Fenton & Neil [14]stipulate that software metrics is a misleading definition to describe all of the different activities undertaken to help understand the software engineering process. They note traditional metrics have not yet proven themselves to be reliable indicators of future code dependability. They highlight that using Bayesian Belief nets as a substitute for the issue of imperfect metrics yields the most success.

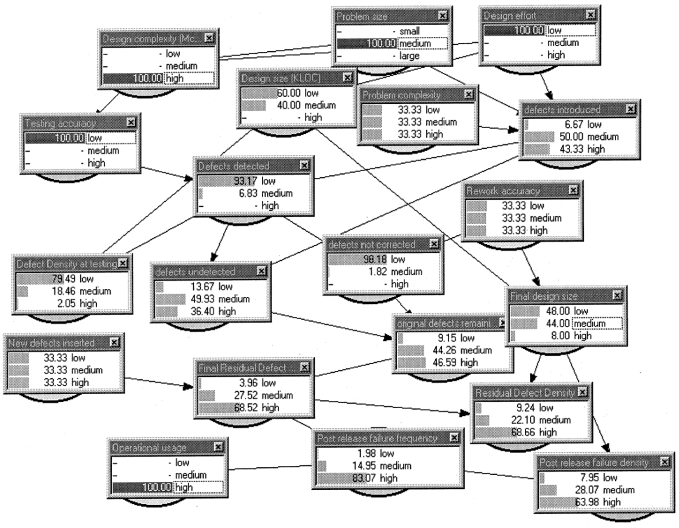

Fig3 Fenton & Neill Diagram demonstrating a potential explanation for the paradox of low density defect pre-release but High density failure post release[15]

Fenton & Neil note that software engineering metrics had been used since the 1960s but focused on LOC and principally ignored the issue of code complexity thus simplifying the engineering process greatly.

The Advancements in algorithms and computing technologies allowed Fenton & Neil to explore the potential of applying Bayesian statistics, a well-established statistical theory to software engineering. Fenton & Neil saw the success of Bayesian belief nets in fields such as Medical Diagnosis where uncertainty is prevalent and successful translated it to avoid the uncertainty of the subjective traditional metrics used in software engineering. A key

[14] http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.63.2683&rep=rep1&type=pdf
[15] http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.63.2683&rep=rep1&type=pdf

advantage of Bayesian Belief Nets is that they intentionally model for ignorance while making explicit assumptions that have previously been obscured from adding auditability to the process, a key point I noted in the Ethics section of the report.

Computational Intelligence

Computational Intelligence can be used to help better understand the software engineering process as it is especially applicable to where a mathematical solution often does not provide the most accurate results. Neural Networks use the brain as a source of inspiration so as to generalise from examples and learn from the past events, similar to human behaviour artificial intelligence and machine learning can give us an insight into trends and commonalities in the investment process. Evolutionary computation is best suited to evaluating a possible solution set in order to identify the most optimal or efficient solution to the problem. [16]

Ethics of Big Data Analytics

The ethics and morality of actions we allow to be determined by a finite set of rules or formulas is an area of huge discussion. In Europe, The General Data Protection Regulation (GDPR) was introduced this year as a constitution for ownership of data within the EU. The EU recognised the value of an individual's data in the modern age and realised the necessity to regulate it.

When we ask what is reasonable surveillance of employees we ought to look at what is currently the practice. Unreasonable use of CCTV in the workplace is illegal[17] and can only be justified in certain instances. I find a tangible link between action monitoring on computers and CCTV in an office. Both means actively track what an individual is doing at a given moment. The fact an employer would then own this footage/data is a worrying as that data can be used to determine an individual's behaviour. Currently by law if employers monitor your internet and mail usage they must inform you of the reasons why they do it.

[16] https://cis.ieee.org/about/what-is-ci

[17] http://www.citizensinformation.ie/en/employment/employment_rights_and_conditions/data_protection_at_work/surveillance_of_electronic_communications_in_the_workplace.html#l6d5f6

The EU has remained at the forefront of ethical protections of Data introducing the "right to be forgotten" and the right of data removal[18]. The right of removal is an incredibly important one and links back to big data analysis in terms that we must now create models that have the potential to "un-learn" or be retrained when data subjects use their right of removal. Companies must also be able to inform the subject what inferences they have made about them from their data and more specifically how it has been used.

Machine Learning and data protection tend not to complement each other, machine learning works best when it has as much data as possible in order to train a model however that same advantage possess an ethical concern to the owner of that data. Research has shown that humans are in fact predictable beings[19] and therefore anyone who holds your data can make basic assumptions about you may or may not do next. assumptions about individuals behaviours are grave and a front to the privacy of the individual

In the case of measuring the software engineering process the company in which the developer works with owns the productivity of that individual, it is by mutual agreement that both the company and the engineer enter into a contract, if it is something an engineer would. Not like to be a part of it is something that such parties could enter into a discussion on how they would like their performance measured.

Measuring the engineering process should be done in a transparent and auditable way

Conclusion

In Conclusion we have identified that there are a multitude of challenges facing the consistent analysis of the software engineering process. The advancement of machine and deep learning technologies may indeed have a massive impact on the area. Companies like Humanyze have begun to collect data which, in the long run may well be able to predict flaws in the programming especially if it can began to understand speech with the aid of Computational intelligence. The ethics of this though is questionable, where do we as a society draw the line between privacy and the interest of increased productivity. This question is one of grave importance and I suspect is likely to be a contentious one for some time to come. In the

[18] https://gdpr-info.eu/issues/right-to-be-forgotten/
[19] https://medium.com/@Si_James_/humans-are-predictable-and-that-s-an-ethical-problem-9a7271ab3e56

interim Bayesian Belief nets serve as a useful tool to deal with subjectivity and the lack of conclusive and actionable data.

Biography

1. https://www.techopedia.com/definition/13296/software-engineering
2. https://www.bcs.org/content/conWebDoc/7895
3. https://www.sei.cmu.edu/researchcapabilities/alwork/display.cfm?customel_datapageid_4050=6520
4. https://deepblue.lib.umich.edu/bitstream/handle/2027.42/43581/11151_2005_Article_BF01024216.pdf?sequence=1&isAllowed=y
5. https://files.ifi.uzh.ch/rerg/amadeus/teaching/seminars/seminar_ws0203/Seminar_3.pdf
6. https://codescene.io/docs/guides/technical/code-churn.html
7. https://go.tasktop.com/rs/246-WDG-185/images/Understanding%20Software%20Development%20Productivity%20from%20the%20Ground%20Up.pdf
8. https://www.techrepublic.com/blog/it-security/the-danger-of-complexity-more-code-more-bugs/
9. https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=5283
10. https://www.win.tue.nl/~wstomv/quotes/humphrey-psp.html
11. https://hackystat.github.io/
12. https://economictimes.indiatimes.com/tech/hardware/protecting-individual-privacy-is-important-ben-waber-cofounder-humanyze/articleshow/64045690.cms
13. https://blog.codinghorror.com/diseconomies-of-scale-and-lines-of-code/
14. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.63.2683&rep=rep1&type=pdf
15. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.63.2683&rep=rep1&type=pdf
16. https://cis.ieee.org/about/what-is-ci
17. http://www.citizensinformation.ie/en/employment/employment_rights_and_conditions/data_protection_at_work/surveillance_of_electronic_communications_in_the_workplace.html#l6d5f6
18. https://gdpr-info.eu/issues/right-to-be-forgotten/
19. https://medium.com/@Si_James_/humans-are-predictable-and-that-s-an-ethical-problem-9a7271ab3e56