Measuring Software Engineering

Student Name: Darragh Murray

Student ID: 18320728

Introduction

This report focuses on the ways in which software engineering processes can be measured and accessed 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.

What is software engineering? Well, according to IEEE, software engineering is 'The application of a systematic, disciplined, quantifiable approach to the development, operation, maintenance and documentation of software; that is, the application of engineering to software.'[i]. In my own words, I would simply define it as a structured approach (although the structured approach may be in and of itself unstructured e.g. Agile) to computer programming.

Methods of Measurement

In a world where speed and reliability are everything, organizations and individuals are constantly searching for ways to improve their own efficiency and code correctness. Software engineering has never been more important than it is now, if an approach to software development can be found that cuts the production time, reduces or eliminates errors in the final product and improves maintainability then it will save an organization or individual countless of hours of work and potentially millions or even billions of dollars. It is therefore no surprise that techniques have been developed that allow us to (to varying degrees of accuracy) measure software engineering.

Lines of Code

One of the oldest and most rudimentary ways to measure software engineering is known as Lines of Code (LOC), Source Lines of Code (SLOC) or even Logical Lines of Code (LLOC). This has been used since the 1960's as an estimation of the complexity of a project.[ii] SLOC involves counting every line of code excluding comments whereas LLOC involves counting every

statement. The accuracy of this metric can be further refined by only counting certain lines of code according to the measurer's choosing.

Due to the simplicity of only counting lines, there are many flaws with this form of measure. For example, contradictory to the logic implied by SLOC, fewer lines of code are actually better, provided the software still provides full functionality, because of the fact that less code is more efficient. If two individuals created a piece of software to calculate the sum of the first 10000 prime numbers, then SLOC would be a terrible way to compare the two programmers if you took more lines to mean better code. Similarly, if software engineers working on a project knew that they were being measured by this metric, then they would purposefully write longer and more inefficient code to make it appear as though they were doing more work.

Hours Spent Programming

Hours spent programming is another simple way of measuring software engineering; however, it too comes with flaws. It can be helpful when trying to identify the 'hardest' part of a project, but apart from that it isn't a reliable form of measurement. A team might be able to compete a task quickly, but that doesn't mean that the task was done properly and error-free, similarly, just because a team spends countless hours on a task doesn't mean that the task was done properly and error-free, it might just mean that the team was very inefficient when it came to solving that particular problem.

Code Coverage

Code coverage is an extremely popular method of measuring software engineering. In a nutshell, it simply measures how many lines of code in a project are being run through tests and can be given as a percentage or number of lines. Using code coverage analysis, development teams can provide reassurance that their programs have been broadly tested for bugs and should be relatively error-free. Unlike the other methods I've mentioned so far, this method doesn't have any pitfalls; how could it be a bad thing that you have tested all your code?

Defect Density

Defect density is the number of defects confirmed in a piece of software/module during a specific period of operation or development divided by the size of the piece of

software/module (usually in lines of code or per thousand lines of code). Essentially, the higher the bug density, the poorer the quality of software. A typical measurement of defect density would be, for example, 5 per KLOC, that is, 5 errors per thousand lines of code. Usually organisations or individuals set their own specific DD thresholds. There are some flaws associated with this measurement, one being that DD doesn't represent the severity of errors, for example one module in a piece of software may have 5 minor bugs in 5000 lines of code and have a DD of 1 per KLOC, whereas another module may have 1 fatal bug in 5000 lines of code and have a DD of 0.2 per KLOC, the latter module will have a lower DD but might not function at all. To offset this disadvantage however is the ease with which DD may be calculated.

Number of Commits

The number of commits is a measurement that only applies to projects that utilise a version control system such as TurtoiseSVN or Git. It is similarly flawed to LOC, but again an easy to compute. Common sense dictates that the size of a commit, number of commits or frequency of commits is not a measure of how impactful a commit is. For example, one developer may commit many times within a day, whereas another developer may choose to only commit at the end of the day; neither suggests the other's work is inferior. If a project wanted developers to commit in smaller, more frequent chunks as to improve the version control readability (i.e. looking through smaller commits is easier than looking through huge commits), then it would be a good measure to see who is complying with that request.

Employee Wellbeing

Software is produced by developers who are, at the end of the day, people. How these people feel can largely dictate their productivity and quality of their work[iii], therefore one of the most important methods of measuring software engineering is measuring the software engineers themselves; finding out how they feel and if they are happy. In our context we can define happiness as being "How positive someone feels that they'll be able to accomplish a goal and feel good about it. It breaks down to their contributions plus their satisfaction in making that contribution.". Measuring this is simple, you simply ask employees how they are feeling but in order for this method to work, the environment requires a work culture in which people feel comfortable sharing such information in the first place.

A more modern approach to measuring employees themselves is by using 'wearables'. These

are devices such as FitBit watches and other smart devices which monitor the movements and vital signs of the people wearing them. With this information one can determine if someone is stressed, anxious, sad, relaxed, happy etc among various other things.

Computational Platforms

A computational platform in the context of software engineering is an environment in which software is developed. Nowadays, there are incredibly sophisticated platforms available to developers, which offer numerous features to enhance productivity.

GitHub

GitHub is perhaps one of the most popular code-hosting (and version control system) websites in the world. The platform offers useful tools to provide an overview of the development lifecycle of a project and various statistics surrounding it. People contributing to a project use commits which are stored and tracked by GitHub. Commits reveal numerous things about an individual including their behavior, competence, work style and productiveness. It also shows how many lines of code have been added/removed.

Pluralsight Flow formerly GitPrime

Pluralsight Flow formerly known as GitPrime is a premium organizational tool that provides productivity analytics software that analyses data from codebases to deliver in-depth reports on the progress of the software development process. The solution uses data from Git-based code repository such as GitLab, GitHub, and BitBucket, to give software teams the basis upon which progress can be benchmarked and give insights to teams which can help them understand which developers are the most effective and productive, which commits are likely to be risky and whether meetings have been productive or not.

Flow attempts to allow developers and teams to “see how much time is spent refactoring legacy code vs new work”, “recognize bottlenecks and remove them” and “get concrete data around commit risk, code churn and impact to better guide discussions”.[iv]

Hackystat

Hackystat is an open-source project devoted to collecting software metrics in an unobtrusive manner. It was created in order to remove the effort required to manually collect data which developers would otherwise have had to have done through documentation and record-keeping. It tracks developers' workflows by attaching to various platforms that developers already use such as editors, build tools or test tools.

There are however ethics concerns arising from the use of Hackystat, with employees expressing discontent over possible privacy violations and the extent to which data is being collected and used.

PROM

PROM is a similar framework to Hackystat in that It tracks developers' workflows by attaching to various platforms that developers already use such as editors, build tools or test tools; however, many of the ethics and privacy violation concerns associated with Hackystat are erased due to the fact that the majority of data generated is only visible to the developers working on the project themselves, promoting a self-analysis approach. Only a summary of the team data is sent to the manager to review.

Jira

Jira, created by Atlassian, is "The #1 software development tool used by agile teams". It allows users to create stories and issues, plan sprints and distribute tasks across a software team. It also improves team performance by providing statistical, real-time data on the project as it develops, allowing a team to self-evaluate its own productivity.[v]

Testrail

Testrail is a test management tool developed by Gurock software. It is a web-based test management tool used by testers, developers and other stake holders to manage, track and organize software testing efforts. It allows developers to create and manage test cases, monitor test results and obtain code coverage statistics. Testrail keeps record of all test case history to maintain transparency and baselines for multiple versions and branches.[vi]

The Personal Software Process – PSP

Personal Software Process (PSP) a structure of working that assists engineers in finding a way to measure and improve the way that they work. It helps them in developing their respective skills at a personal level and the way of doing planning, estimations against the plans.

PSP aims to help software engineers improve their approximating and planning skills, make promises that can be fulfilled, manage the standards of their projects and reduce the number of faults and imperfections in their work.

PSP maintains that developers must measure and count the time they spend on different activities during development, as well as create a plan before beginning development to ensure that no effort is wasted on unimportant activities and to ultimately avoid a poor and unsatisfactory product.

There are four levels involved in PSP:

PSP 0: The first level of Personal Software Process, PSP 0 includes Personal measurement , basic size measures, coding standards.

PSP 1: This level includes the planning of time and scheduling .

PSP 2: This level introduces the personal quality management ,design and code reviews.

PSP 3: The last level of the Personal Software Process is for the Personal process evolution.

Algorithmic Approaches

Halstead's Complexity Measures

In 1977, Maurice Howard Halstead an algorithmic approach to measuring software complexity. The algorithm was developed to calculate the complexity of source code through the counting of the operators and operands in the module. According to Halstead, "A computer program is an implementation of an algorithm considered to be a collection of tokens which can be classified as either operators or operands". His algorithm ascertains a variety of measurements such as vocabulary, testing time, size, errors, difficulty, and efforts for C/C++/java source code. His metrics are now included in a variety of commercial code reviewing tools such as Microsoft office applications.

The algorithm takes the following inputs:

- n_1 : Number of distinct operators.
- n_2 : Number of distinct operands.
- N_1 : Number of operator instances.
- N_2 : Number of operand instances.

Using this information, it can then calculate the following measures:

- n, the vocabulary, which is $n_1 + n_2$
- N, the size, which is $N_2 + N_2$
- D, the difficulty, which is $n_1/2 * N_2/n_2$
- V, the volume, which is $N * \log2\ n$
- B, errors, which is $V / 3000$
- E, the effort, which is $V * D$
- T, testing time, which is $E / 18$

Bayesian Belief Network

A Bayesian network (also known as a Bayes network, belief network, or decision network) is a probabilistic graphical model that represents a set of variables and their conditional dependencies via a directed acyclic graph.

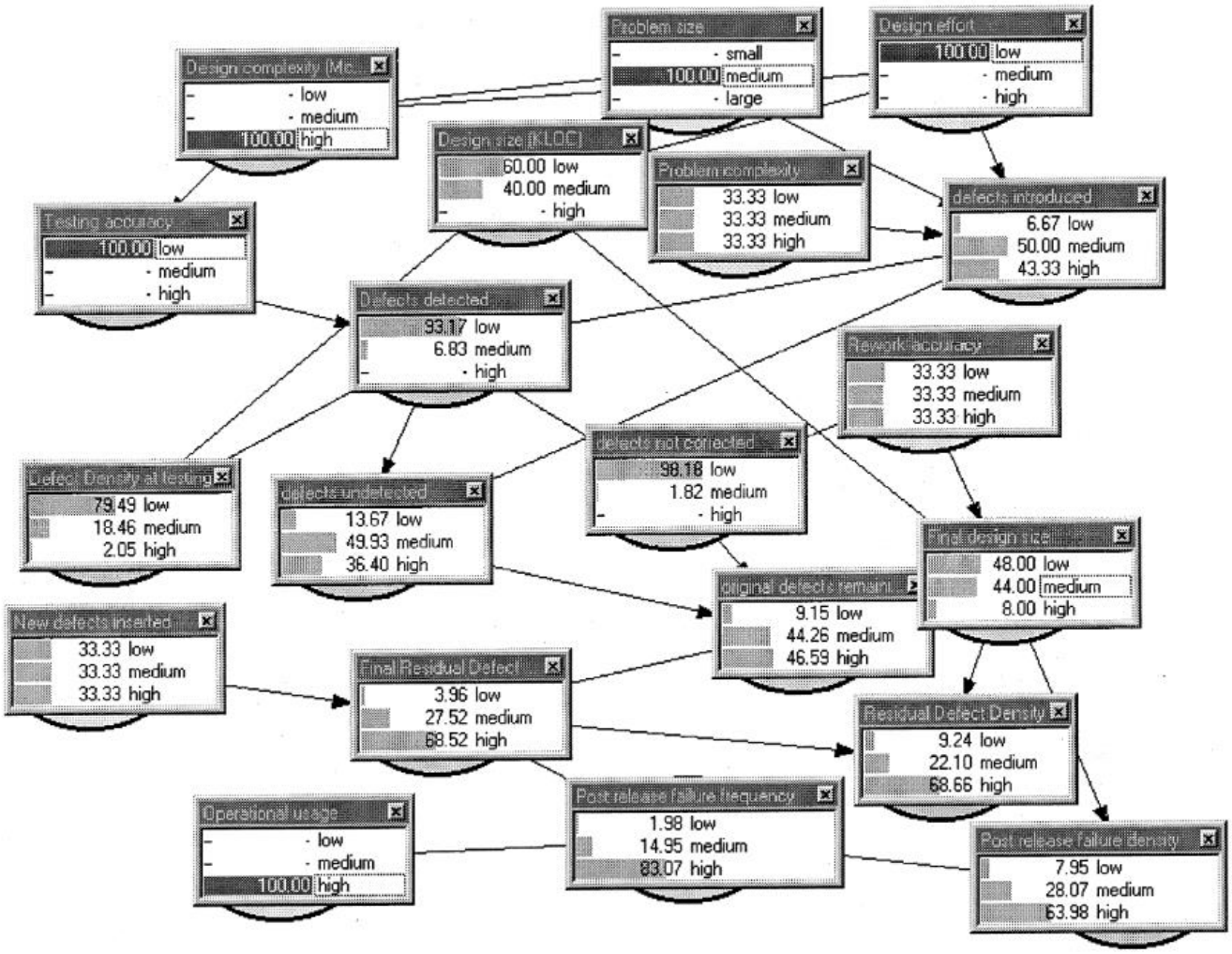
The benefits of using BBNs as stated by Fenton (1999) are:

- Explicit modelling of 'ignorance' and uncertainty
- Makes explicit assumptions that were previously hidden thereby adding transparency to a project
- An intuitive graphical format makes it easy to understand
- Ability to forecast missing data
- Rigorous, mathematical semantics for the model

- No need to do any of the complex Bayesian calculations, since tools like Hugin are available to do this

BBNs are well established in industry, according to Fenton (1999) “To date BBNs have proven useful in practical applications such as medical diagnosis and diagnosis of mechanical failures. Their most celebrated recent use has been by Microsoft where BBNs underlie the help wizards in Microsoft Office.”[viii]

Below is a graph of a BBN taken from Fenton’s book:



Ethics Concerns

Wearables, software that latches onto the tools you use to record your every click…where do we draw the line and how do we ensure that this data isn't improperly used?

Monitoring efficiency and effectiveness in the workplace will definitely help to improve the aforementioned qualities, but what exactly is the difference between monitoring and stalking? Do employees know exactly how they are being monitored and are they comfortable with the level of monitoring occurring? Irish law answers a couple of these questions. In Ireland, you must be given details of the monitoring, such as: who is monitoring you; what they are specifically monitoring; how are they monitoring you and when are they monitoring you.[ix] If an employer wishes to monitor your internet use or emails, it must be necessary, legitimate and proportionate to the perceived risk of you doing something bad through the use of internet e.g. leaking confidential details.

Personally, I believe that in many cases monitoring is essential, not for productivity, but for security, particularly in such a situation where sensitive information is involved such as in hospitals or government buildings. There are of course other situations. Take for example a software company that forces its employees to download computer monitoring software on their home PC during the coronavirus pandemic, to ensure that the employee is working during work hours. Whilst I understand the concern of the employer, I equally understand the concern of the employee, who would wonder (as any of us would), is the data that is now visible to my employer safe? I believe that if an organization is going to impose the monitoring of peoples' devices, particularly personal devices, then that organization must also demonstrate exactly how that data is going to be used and how it is going to be kept safe. Thankfully under Irish law, an employee can request the data than an employer has gathered on said employee and the employer must respond within 1 month. In the case that an employer isn't willing to comply, a complaint may be sent to the Data Protection Commission.

Wearable technology also poses a problem; these devices can monitor many things including your heart rate, location, things you say, movements you make. Personally, I would rather choose to tell others how I feel than to have a device that I'm forced to wear do it for me. If information is being collected unnecessarily employees with concerns may begin to resent their employer, and this could backfire on the entire purpose of a wearable which is to enhance productivity, as it has been shown time and time again that people do not perform as well or efficiently when they do not feel happy or comfortable.

The best way to approach the monitoring of employees is to ask them what they are comfortable with and be extremely transparent in how they are monitored.

Conclusion

With an array of ways to measure the software engineering process, one must exercise caution when analyzing the result of each; there are many different aspects to software engineering and no single measurement will capture all aspects equally.

Furthermore, with how technologically advanced we have become, there are now methods of observation and monitoring that to some cross the line between monitoring and stalking / spying. It is more important now than ever for legislation to be quickly introduced to clearly state what a company or organization can and can't do in relation to monitoring employees.

Measuring software engineering is a game of trade-offs, if you monitor people too much, they become uncomfortable and unproductive, if you rely on specific metrics too much, you will not see the big picture, it's just a matter of fine-tuning and finding out what works best for your specific project.

Bibliography

[i] https://www.tutorialspoint.com/software_engineering/software_engineering_overview.htm#:~:text=%20IEEE%20defines%20software%20engineering%20as%3A%20%201,approaches%20as%20in%20the%20above%20statement.%20More%20

[ii] Fenton, N. E., and Martin, N. (1999) "Software metrics: successes, failures and new directions." Journal of Systems and Software 47.2 pp. 149-157.

[iii] Oswald, A.J., 1997. Happiness and economic performance. The economic journal, 107(445), pp.1815-1831.

[iv] https://www.pluralsight.com/product/flow?utm_term=&aid=7014Q0000022a2tQAA&promo=&utm_source=branded&utm_medium=digital_paid_search_bing&utm_campaign=Bing_EMEA_Brand_Flow_E&utm_content=&utm_term=&msclkid=e378ea232490109f734eb5e32440131c

[v] https://www.atlassian.com/software/jira

[vi] https://www.gurock.com/testrail/tour/modern-test-management

[vii] https://en.wikipedia.org/wiki/Halstead_complexity_measures

[viii] Fenton, N. E., and Martin, N. (1999) "Software metrics: successes, failures and new directions." Journal of Systems and Software 47.2 pp. 149-157.

[ix] https://www.citizensinformation.ie/en/employment/employment_rights_and_conditions/data_protection_at_work/surveillance_of_electronic_communications_in_the_workplace.html

General Sources:

- https://stackify.com/measuring-software-development-productivity/
- https://stackify.com/track-software-metrics/
- https://www.tutorialspoint.com/index.htm
- https://www.jmoses.co/2019/07/08/software-engineering-manager-guide-measuring-performance.html