MEASURING SOFTWARE ENGINEERING

Nathan Carney

18326931 JS MSISS

Table of Contents

1. Introduction

1.1 What is Software Engineering?

Software Engineering is "a process of analysing user requirements and then designing, building, and testing software applications which will satisfy those requirements" [1]. A software engineer's role is to optimise the software development process so that software is satisfactory for solving a user's problem, and coming up with ways in which to improve upon this process.

1.2 Why should Software Engineering be measured?

The famous words of Lord Kelvin are apt for answering this question; "If you can't measure it, you can't improve it".

Efficient and malleable software acts as the bedrock of any tech companies' success. Companies such as FANG (Facebook, Amazon, Netflix, Google), whose products and services are embodied by this idea, occupy a significantly large proportion of the world's economy, in terms of both popularity and profit.

Through measurement, companies such as those mentioned above can (and do) assess productivity levels in various departments and teams so as to spot potential avenues for change. It also gives managers and those in charge of several groups of developers a clearer picture of who is performing well and who is not. Measurement also presents developers with indicators of software performance, and quality of tests.

The analysis of employee and team metrics, and the deployment of computational and algorithmic tools to assess these metrics is, hence, an undoubtedly lucrative and worthwhile task. However, it also presents both qualitative and quantitative challenges and raises some moral questions. This will serve as the basis of this report, as I will examine the measurement of Software Engineering in the world today.

2. Measurable Data

In order to assess how effectively software engineering and development can be measured, the possible metrics used in measurement and their viability must be examined. A metric is "a quantifiable measure that is used to track and assess the status of a specific process" [2]. In this case, the process under examination is software development. How can we decide whether or not an employee's contribution to a project has been satisfactory, or how effective their code is in handling the user-defined problem? How can we assess the performance of a team of developers, or the quality of a final product?

[1] https://www.guru99.com/what-is-software-engineering.html
[2] https://www.klipfolio.com/blog/kpi-metric-measure

2.1 LOC & Commit Count

With this approach, employee performance is measured by how many lines of code they have written (LOC), or how many commits they have performed to a given Version Control application, for instance GitHub or BitBucket. Using this metric, an employee may be incentivized to simply write as many lines of code possible to solve even the most menial and trivial problems. They may also commit insignificant changes to the code base to appear more productive.

As Bill Gates said,

"Measuring programming progress by lines of code is like measuring aircraft building progress by weight".

A software engineer can occupy many distinct roles within an organization, as there are many distinct aspects of software development. For instance, testing, maintenance and design. An engineer tasked with test implementation will create test cases which differ greatly in length to the code base being tested.

When applying the metric of a commit count on, for example, GitHub, one engineer may commit bodies of code which are trivial or not relevant to the overall project, while another may have fewer commits yet more substantial code when attributed to the overall project.

Upon assessing the performance of the final product with this metric, the culmination of these issues is revealed. The number of lines of a software project is almost completely arbitrary. If the software provides the functionality sought after by the client, this metric may be irrelevant.

Hence, this metric, while the simplest to collect, presents many glaring inconsistencies. When used to measure the effectiveness of a code base, it can reward either longer or shorter solutions, when many sections of code may require different lengths for different tasks. When used to measure the effectiveness of a single employee, it can't differentiate between different approaches to problem-solving, or between different roles.

2.2 Team Metrics

Most of a software engineer's career will be spent working in a team. Teams are integral to the field of software engineering, and without them the creation of software would take an exponentially longer amount of time. Travis Kimmel, CEO and Founder of GitPrime, outlines in his article "5 Developer Metrics Every Software Manager Should Care About"[3], the essential aspects of the software engineering process which managers should consider when measuring the productivity of their teams. He distills these down into 5 key metrics: Lead Time, Code Churn, Impact, Active Days, and Efficiency.

[3] https://www.pluralsight.com/blog/teams/5-developer-metrics-every-software-manager-should-care-about

2.2.1 Lead Time

Lead time is essentially the length of time it takes for a project to be completed and made available to the customer. Kimmel outlines in the article how the past lead time of a team's projects can be used to measure how long they may take with future projects of a similar scope. The following example is provided:

"assume that 50 percent of similar feature request had a lead time of two weeks or less, and 90 percent of these projects had a lead time of one month or less. You could confidently provide a lead time of one month for the current project"

This metric undoubtedly gives managers an effective way to estimate future lead times, however it could be argued that the 'scope' of different projects may prove challenging to compare. This would mean that estimates could be unreliable.

2.2.2 Code Churn

The idea of code churn, as expressed in the article, is summarised as follows:

"Code Churn is the percentage of a developer's own code representing an edit to their own recent work. It's typically measured as lines of code (LOC) that were modified, added and deleted over a short period of time such as a few weeks"

This is an example of how the LOC metric can be applied to a more useful metric. By analysing the rate at which a developer within a team is adjusting the same LOC, and the amount of LOC they are adjusting, a manager can gain an important insight into how the developer is dealing with problems in a project.

For instance, if a developer has been constantly deleting the same section of a code base and re-doing it, a manager could discover that the developer is having a lot of trouble with this part of the project and needs additional help, or needs a more experienced developer to help them.

This metric could run into some issues when considering the reasons behind frequent changes to their code. They could simply be responding to a change in client requirements.

2.2.3 Impact

Impact is a measurement of the extent that code changes affect the project being undertaken. As mentioned in the article, this can be visualised as a comparison of the number of LOC adjusted in a program to the number of files affected after the change. The bigger the change, the larger the value of impact. This value can be compared to previous values to determine the extent of the current change set's impact.

2.2.4 Active Days

Active days are the number of days which a developer has contributed to a project. This can be calculated from the developer's version control log-ins, for instance. When these metrics are used with the broader context of team meetings and distractions faced by any given

developer taken into account, a manager can get a broader picture of what is effecting an employee's productivity or commitment.

An issue with this metric is the possible discrepancy between hours worked and level of competency. A highly skilled and experienced worker will need to spend much less time tackling a problem than a newer, low-skilled developer.

2.2.5 Efficiency

As stated in Kimmel's article, efficiency takes a slightly different meaning than the traditional definition; where it's described as " the percentage of an engineer's contributed code that's productive, which generally involves balancing coding output against the code's longevity… The higher the efficiency rate, the longer that code is providing business value. A high churn rate reduces it".

An employee's efficiency is hence characterised as the amount of code they have written which has remained in use and remained of value for a long period of time. An issue with this metric could occur if the code base being used is not fully optimised, but is being used because of the high costs involved in changing the code for the business.

2.3 Employee Happiness

When happiness can be measured, it gives a manager an insightful indicator of developer's progress throughout the development process. Studies have shown that when a team is happy, their productivity and creativity sky-rocket, by 37% and 300% respectively[4]. Companies such as Hitachi have been performing research and tests in this area, and new ways to quantify happiness have been innovated.

Despite its usefulness if effectively measured, happiness is inherently subjective. Caution must be exercised when using this as a metric. When, perhaps, a mixture of sensors and 'happiness' questionnaires are used to give an idea of a person's happiness, results can be misleading and depend heavily on the quality of data analysis performed and questions posed to individuals involved.

2.4 Software Reliability

An obvious measure of the software engineering process and developer productivity is how 'reliable' the software project is once completed. Ian Summerville, writing for Hackernoon.com, defines reliability as "the probability of failure-free operation for a specified time in a specified environment for a specific purpose"[5]. The higher the probability, the more reliable the software, at least theoretically. This can be calculated by an automated running of tests and debugging.

An issue with reliability as a metric lies in its subjective nature. While we can quantify the chances of aspects of the software failing to run as necessary, this could mean different things for separate pieces of software. If aspects of two distinct projects have the same failure rate, this does not necessarily signify the same reliability in both projects. One project could

[4] http://www.hitachi.com/rev/pdf/2015/r2015_08_116.pdf
[5] https://hackernoon.com/software-reliability-pt-1-what-is-it-why-developers-should-care-f2a71274772c

perform its most primary function relatively well every time it is run, yet have a large number of failures in more tertiary parts of the code, while another project's primary function may fail consistently and result in the rest of the system not working.

2.5 Dynamical Influence in Human Interaction

As has been stated previously, teams are integral to software engineering. Measuring their progress and results is a priority for any company hoping to improve upon its success. Many studies have been carried out on team interaction and its implications for project completion. One such study[6], conducted in UC San Diego, reviews the 'influence model', "which utilizes independent time series to estimate how much the state of one actor affects the state of another actor in the system", and expands on it to propose a 'dynamical influence model' which changes the model from static to dynamic.

This is undoubtedly useful when analysing the ongoing progress of a development team, as more efficient and performative employees can be identified, as well as those employees who are not speaking up or contributing.

The paper defines 'influence' as the idea that "an outcome in one entity can cause an outcome in another entity". Using the old model and the proposed model, researchers found they could identify influence patterns in group discussions and predict turn-taking in group discussions, respectively.

One caveat to these models, as is also stated in the above paper, is that influence is incredibly hard to successfully model mathematically, having been attempted by both data and social scientists alike countless times. Hence, when interpreting results one must be careful not to overestimate their validity. Additionally, in various applications of these models employees had to be fitted with sensory devices which recorded their discussions and interactions. This may be a hurdle to effectively using influence as a metric if employees express privacy concerns.

2.6 Implementing Video Game Theory into Software Development

An interesting approach to measuring parts of the Software Engineering Process, proposed by Erick B. Passos, Danilo B. Medeiros, Pedro A. S. Neto and Esteban W. G. Clua [7], is to design metrics based on the core ideas of video game development. Metrics such as projects completed, effective testing, and retrospective software reliability can be grouped together to grant 'achievements' to developers to make the process of Software Engineering more "fun". Developers can aim to complete more achievements and write more effective code than their peers, adding a layer of competitiveness and accomplishment where there may not have been before.

This added layer of competitiveness, however, may result in a reduction in co-operation in employees, as they may wish to focus on gaining their own achievements rather than helping others achieve theirs, in order to appear more productive to their superiors.

---

6 https://dspace.mit.edu/bitstream/handle/1721.1/92443/Pentland_Modeling%20dynamical.pdf?sequence=1&isAllowed=y

7 http://www.sbgames.org/sbgames2011/proceedings/sbgames/papers/comp/full/30-91552_2.pdf

3. Computational Platforms

There exist numerous platforms today which can be used to obtain the above metrics. However, finding any platform capable of accurately measuring all or even most of these metrics is quite difficult. As will be examined below, there is somewhat of a dilemma faced by software developers when deciding which platforms to use, or whether to use metric-collection platforms at all. Where a platform lacks automation in the collection of data, human error and 'collection fatigue' is prevalent. Where automation removes human error, computational and contextual errors arise frequently.

To begin, the three 'generations' of metric collection toolkits as set out in the paper 'Beyond the Personal Software Process: Metrics collection and analysis for the differently disciplined'[8] will be analyzed.

More modern platforms will subsequently be examined.

3.1 Traditional Toolkits

3.1.1 PSP

The PSP (Personal Software Process) is a method of keeping track of personal progress and issues encountered while developing software. The framework was first set out in 'A Discipline for Software Engineering' by WS Humphrey in 1995. This paper and its ramifications are discussed in a subsequent review of metrics collection by lecturers in the University of Hawaii. According to this review, the primary goals of the PSP framework are to "improve project estimation and quality assurance". Until the mid 90's, this process of collecting metrics was handled at an industrial level by employees hired specifically for the purpose of collecting department metrics, but Humphrey's paper proposed a way for developers to use it to maintain their own metrics. It requires developers to analyse projects with regard to aspects such as size and time taken to complete, so as to give a better indication for the estimated time taken to complete future projects. Developers must print out forms to keep track of this information while working on software, along with defect information and overall effort involved.

The results of implementing PSP have yielded impressive results in software engineering students [8] and increased the overall quality of their work. The process has many qualities, including the ease involved in adapting the framework to any new project, given its universal and standardised properties.

However, the idea of 'context switching' presented itself as a barrier to more regular adoption of this framework after students graduated. 'Context switching' refers to the extra effort needed to move from completing software projects to performing self-analysis. Due to the paperwork and numerous forms needed to effectively keep up this self-analysis it was found that, at least anecdotally, students rarely continued to use PSP post-academia[8]. Additionally, it is clear that this process is prone to human error, with the developer themselves filling in their own forms.

[8] https://ieeexplore-ieee-org.elib.tcd.ie/stamp/stamp.jsp?tp=&arnumber=1201249&tag=1

Hence, PSP in its original form proved unsustainable as students moved into the working world.

3.1.2 Leap

To overcome these barriers of adoption, the above lecturers in the University of Hawaii started to develop a more automated approach to metric-collecting in 1998, titled 'Leap'. 'Leap' would significantly reduce overhead costs involved with metric-collecting by eliminating the need to constantly fill out forms logging effort and compiler errors. The 'Leap' Toolkit was formatted as a dashboard which contained dialog boxes, into which users could input relevant defect and effort information. This information could then be displayed back to the user upon request to get a more visual feedback of their recorded metrics. Hence, the process of personally reviewing recorded metrics was eliminated and automated. The dashboard also provides users with the ability to perform linear regression on metric information. This dashboard also displays metrics such as churn, LOC and Commit count.

The issue of context switching remained, however, as while "adoption improved slightly"[8], many students did not use the toolkit after college. It was found that, again, students and graduates alike found it difficult to constantly switch back and forth between writing software and logging their results and errors. Thus, despite the highly increased automative capacity of the Leap toolkit, it was similar in unpopularity as compared with PSP.

3.1.3 Hackystat

The Hackystat toolkit was the subsequent answer to the issue of automation. It begun development in 2001 by the same group of lecturers mentioned previously. Their aim was to eliminate context switching and reduce overheads. In this they succeeded. The toolkit required the user to sign up with their email address and attach sensors to their development tools. These sensors collected metrics automatically by sending them to a host server and sending summaries of these metrics back to the user via email. The toolkit records effort, progress and defects in 5-minute intervals by keeping track of a local directory or package. A user can view their daily progress in their 'Daily Diary'.

There exists a number of issues with the Hackystat toolkit. Firstly, it is better optimised for certain languages over others. As stated by the creators, "Emacs is easy to integrate, Notepad is not" [8]. The toolkit was developed using JBuilder and works well with Emacs and Java, but less so with other languages. This creates a clear barrier to adoption. Additionally, the process requires the installation of sensors and places requirements on developer hardware.

Hence, similar to PSP and Leap, Hackystat has also yet to see an emergence outside of the world of academia. As of the writing of this report, Hackystat is no longer under active development [9].

3.2 Modern Employee Measurement Tools

Tom Gorski, of Software as a Service, in his article 'How to Find the Best Software for Measuring Employee Productivity'[10], outlines the best platforms available today for

[9] https://hackystat.github.io/
[10] https://www.saasgenius.com/blog/how-find-best-software-measuring-employee-productivity

measuring developer productivity from the perspective of a team manager. These platforms are distinctively more ‘modern’ than those outlined above. They are currently in use by developers today in the working world and are being used by managers to assess employee metrics and productivity.

In the article, Tom advises every team manager to question why they are seeking to calculate a given metric, and recommends the following framework to advise a manager’s decision to adopt a certain productivity measuring tool:

- The tool must inform decisions
- The tool must be based on accurate and measurable data
- The tool should motivate employees and improve performance

These tools are listed below:

3.2.1 Project Management

Project management tools are used to ensure employees are keeping on top of their work and kept accountable for completing tasks. Their usual premise is setting out a pre-defined project which has a specific deadline as well as its associated tasks, and visualizing these in an easily digestible manner. Tools such as Trello, Asana and Monday are particularly useful for undertaking this kind of work.

The primary issue faced by both managers and developers when using these tools is that they work best with predictable projects and those with a repetitive framework. When “less predictable” projects arise, these tools tend to bog down the user with unnecessary work, as tasks will need to be entered with sufficient detail.

3.2.2 Time Tracking

Time tracking presents an interesting decision for managers when keeping track of employee progress. With the current and ongoing COVID-19 pandemic resulting in an exponential increase in employees working from home, time tracking can give managers access to how their employees and teams are truly allocating their time and resources at home. The same can be said for free-lance developers and remote, contracted workers / developers whose hours can be difficult to measure for those hiring them.

Two types of time tracking can be used to measure a worker’s productivity: automated time tracking and user-controlled time tracking. The former can unobtrusively log a developer’s time spent on any given application or project. The latter allows the employee to enter their hours worked themselves. The main positive associated with this measurement is the fact that the data collected is hard to misinterpret when collected through automation. Employers can get a reliable sense of how their employees are working. Tools for collecting this metric include: TimeDoctor, Toggl, and GetHarvest.
Significant issues can also be found in using these tools to gather employee information. If the employee is entering their own hours, there is a very real risk of dishonesty and unreliable data. If the hours are being logged automatically, this can place unnecessary pressure on developers to appear as though they are working, even if they do not take a lot of time with a

given project. Additionally, employees can feel as though their managers do not have faith in them to get the job done on their own.

3.2.3 Team Productivity and Collaboration

When a team is assigned any project, it is essential that they all keep each other updated and in the loop on everyone's progression. Team productivity and collaboration tools are built to digitize and streamline this process. Teamweek, Actioned, Teambook, iDoneThis, all are prominent examples of tools used in the modern workplace, by software engineers and those in many other disciplines. If the onset of COVID-19 has shown one thing, it is the importance of effectively handling remote, team-based collaboration.

For instance, an application like iDoneThis shows each team member's completed tasks and in-progress tasks. This enables all team members to gain an informative picture of where they stand with regard to project completion.

The input for these tools is completed by humans, hence presenting an increased risk of human error. One team member's interpretation of a 'completed' task may differ greatly to their colleague's.

3.2.4 Results Dashboard

A dashboard is a particularly useful tool for managers when reviewing team results in their department. For instance a team specializing in web development can go to their manager after implementing a new design for the company site with data relating to increased site visits and increased sales. Reporting tools such as Sisense and Zoho provide simple and digestible templates to display data.

A concern that could be expressed with this tool is its favorability towards smaller teams. As teams grow in size, each employee's contribution to a given project is diluted, and this can result in an increased sense of insignificance relating to project results from the perspective of the individual employee.

4. Algorithmic Approaches

There remains an important question when examining the measurement of software engineering, which is how to do so via automated means through algorithms. Can a sufficient, logically sound algorithm be formulated to calculate many of the metrics listed in previous sections? Following research of a handful of these algorithms, the answer is unclear. There are certainly accurate methods of analysis and reliable algorithms that can be conducted and performed on a code base to determine its maintainability, reliability and functionality. However, where the human factor in software development is concerned, certain metrics can be hard to quantify, though significant strides are being made by many companies to achieve this.

Below, algorithmic approaches to measuring certain aspects of code and software will be discussed. The analytics of the people working on the software will then be examined.

4.1 The Halstead Approach

The Halstead approach to measuring software complexity focuses on the number of operators and operands present in a code base. The method can be used for any programming language[11]. An operator in a programming language tells the computer what action it would like it to perform [12]. They can consist of simple '+', '-' and '*' symbols, but also 'if', 'else' and 'for' statements. Operands are the values being manipulated by the operator. These are usually common variables found in any program.

By reviewing the source code and counting these values, the Halstead complexity of a program can be established. This complexity gives any developer a clear idea of a program's testability and the difficulty they may encounter when maintaining the code after it has been completed.

The following calculations are gathered using this algorithm;

n1 = Number of distinct operators.

n2 = Number of distinct operands.

N1 = Total number of occurrences of operators.

N2 = Total number of occurrences of operands.

Figure 1

To extrapolate the following components of a code base:

Metric	Meaning	Formula
n	Vocabulary	$n_1 + n_2$
N	Size	$N_2 + N_2$
V	Volume	$N * \log2\ n$
D	Difficulty	$n_1/2 * N_2/n_2$
E	Effort	$V * D$
B	Errors	$V / 3000$
T	Testing time	E / k

Figure 2

[11] https://www.geeksforgeeks.org/software-engineering-halsteads-software-metrics/
[12] https://users.cs.cf.ac.uk/Dave.Marshall/PERL/node32.html

This algorithmic approach has its benefits in analyzing software. The algorithm is relatively simple to perform on a program, it can be used universally in any programming language, and it gives developers a strong idea of the quality of their software. As stated previously, efficient and malleable software is essential in the modern world.

However, this approach comes with drawbacks. The use of the Halstead method is restricted to a finished project. When 'measuring software engineering', it is useful to be able to do so while in the process of completing the software.

4.2 Cyclomatic Complexity

Cyclomatic complexity was developed by Sir Thomas McCabe in 1976. The overall aim of this software measure was to determine the number of 'choices' within source code which in turn enables developers to reduce overall program complexity and ascertain an estimate of the number of test cases required[13].

This complexity measure formats the overall program as a control flow diagram, i.e a graph consisting of nodes and edges[14]. Each 'path' through the program is represented by its own node and edge. The diagram below illustrates this idea:

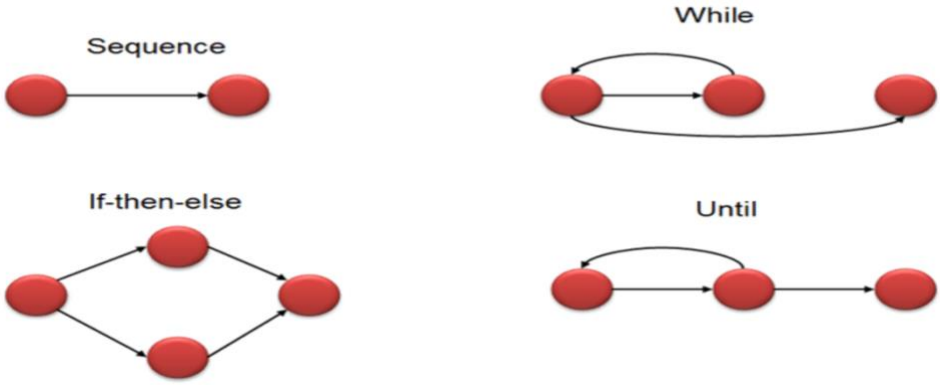

Figure 3

From the visual representation, the following algorithm can be used to calculate cyclomatic complexity:

```
V(G) = E - N + 2
```

Where,

E - Number of edges

N - Number of Nodes

```
V (G) = P + 1
```

Where P = Number of predicate nodes (node that contains condition)

Figure 4

[13] https://www.perforce.com/blog/qac/what-cyclomatic-complexity
[14] https://www.guru99.com/cyclomatic-complexity.html

The higher the value obtained from the above formulae, the more complex the code. Typically, complexity ranges from 0 to 40. A result of 10 means the code is easily tested, 40 signifies that the code is virtually untestable. Hence, developers should aim to reduce the overall number of 'independent paths' within their program in order to require less test cases.

A disadvantage of solely focusing on complexity measures such as those listed above is that these do not account for defect detection or uncertainty.

4.3 Bayesian Belief Networks

In the case study 'Software Metrics: Successes, Failures and New Directions'[15] by Norman E. Fenton and Martin Neil, Bayesian Belief Networks are proposed as an approach to software modelling and measurement. In their investigations they found that the ability of Bayesian probability to account for uncertainty made it stand out from other algorithms and models for measuring the software process (this being referenced in the paper as the DATUM project).

Bayesian Belief Networks (BBN's) are used in other areas, such as Microsoft Office's help wizards and medical diagnosis. BBNs are graphical networks, accompanied by a set of corresponding probability tables. A network is comprised of nodes and arcs. In these networks, the nodes represent 'unpredictable' variables such as undetected defects in a program, and the arcs represent "the causal/relevance relationships between the variables"[15].

Each node has its own corresponding probability table outlining each possible state of that variable. The diagram below shows an extract from the case study performed by Fenton and Neil, displaying various nodes and the relationships between them. This diagram illustrates the "defects detection and insertion process":

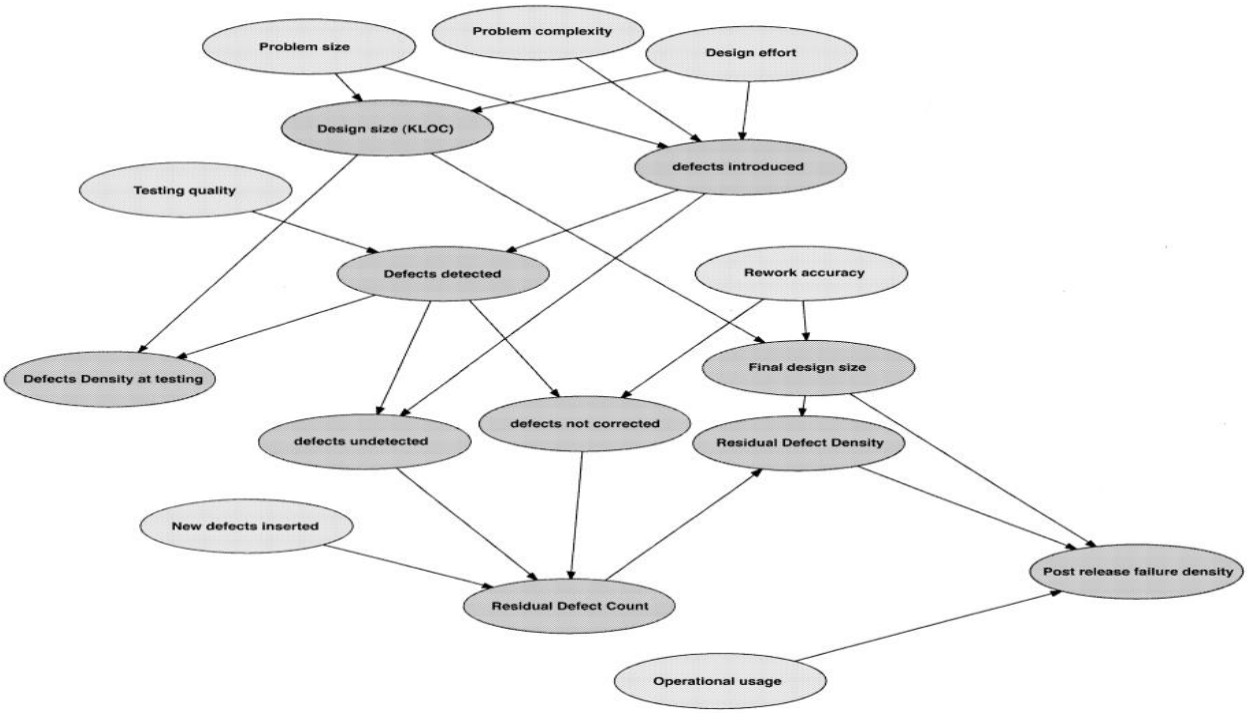

Figure 5

[15] https://eecs.qmul.ac.uk/~norman/papers/jss_1999.pdf

The above paper concluded that BBNs has received "highly favourable reviews" and is being used on "real projects". It must be noted that this paper was published in 1999, and clearly software has evolved considerably since then.

Yet, a recent article[16] published in 2019 (20 years later) still describes BBNs as:

"a useful tool to visualize the probabilistic model for a domain, review all of the relationships between the random variables, and reason about causal probabilities for scenarios given available evidence"

In the same article, the author mentions recent libraries in Python such as the PyMC3 library, which give developers relatively easy access to both the visualization and functionality of this algorithmic approach.

4.4 People Analytics

Thus far, algorithmic approaches focusing on software have been examined. This represents only one part of the software engineering process, as the engineers writing the software and approaches to measuring their own quality must also be discussed.

The analytics of software developers (and employees of any discipline) has taken many forms in recent years.

For instance, using 'bots' or AI to analyse employee behaviour has become the selling point of Sapience Analytics. Sapience Analytics is a company providing solutions to firms that want to better understand and improve employee work patterns. Their current product, Sapience Vue, uses a bot which they have entitled 'Len' to incrementally report data about a given employee, sometimes every 15 seconds[17]. Like many workplace analytics products, Sapience Vue has become quite a topical and relevant tool in modern times. Due to the COVID-19 pandemic, many employees are now working from home, away from the observational eye of their bosses. Tools like these keep these employees accountable and motivated to be as productive as they are in the office.

People analytics is not unique to Sapience. The approach to measuring employee productivity has become a staple of major tech firms, such as Google. In an article by Dr. John Sullivan for Talent Management and HR[18] 'How Google is using People Analytics to Completely Reinvent HR', it is argued that in any company "accurate people management decisions are the most important and impactful decisions that a firm can make". The article outlines how in Google these decisions are highly data-driven, and their "people analytics team" have stressed that their primary goal is to "bring the same level of rigor to people-decisions that we do to engineering decisions". One particular aspect of this strategy is Google's PiLab, a subgroup unique to Google. The group carries out focused experiments to decide on the best approaches for making the work environment productive and for managing employees.

Two algorithms which are being used by the company in their 'people operations' are their retention algorithm and hiring algorithm. The retention algorithm is designed to alert team

[16] https://machinelearningmastery.com/introduction-to-bayesian-belief-networks/
[17] https://sapienceanalytics.com/product-overview/faqs/

[18] tlnt.com/how-google-is-using-people-analytics-to-completely-reinvent-hr/

leaders when employees are close to leaving the company. The hiring algorithm is built to predict which employees will be most successful and valuable to the company upon hiring them. After implementing this hiring algorithm, Google reported a 'miss rate' (promising candidates who slipped through the cracks) of only 1.5%.

Clearly, algorithmic approaches to measuring employee (and potential employee) productivity are now commonplace in organisations, yielding consistently impressive results.

However, these impressive results come at a cost. One glaring issue has yet to be discussed with the metrics, platforms and algorithmic approaches surrounding the measurement of the software engineering process: ethical concerns.

5. Ethical Concerns

I am of the opinion that this type of analysis must be handled very carefully. Extensive measures must be taken to ensure that data is collected in the right way. Both using the correct methods and keeping the wellbeing of those involved in the data's collection at the forefront of decision-making should be prioritised equally when measuring any component in the software engineering process and beyond. It seems as though all too regularly the latter priority is forgotten in this process, for the sake of increased productivity and higher profits.

5.1 Accountability and Transparency

Employees must be fully aware of how and for what purpose their data is being collected, and they must give permission to do so. Strides have been made to ensure this notion is upheld. For instance, the passing of GDPR regulations in the EU has made all citizens within the EU at least partially more aware of the extent to which companies collect data, and the frequency at which it occurs. I believe that this is vital. There must be accountability present for any entity, employer or otherwise, to disclose clearly and fairly information such as this. By looking back to 2016, breaches of this principle are clear. Data analytics company Cambridge Analytica knowingly collected personal information on millions of US voters and sold this information for a third party's political advancement. This precedent had to be stopped in its tracks and prevented from becoming commonplace in society, which thankfully (or, hopefully) happened.

If an employer of mine unethically shared information collected about my work as a software engineer without my knowledge or consent, I know that I would not stand for it.

5.2 Purpose

I see little to no problem in the measurement of productivity. As was quoted at the beginning of this report, "If you can't measure it, you can't improve it". However the underlying goal of this kind of analytics should be moralistic. Employees should not be defined by their metrics once collected, nor cast aside in the pursuit of greater perceived performance. Their metrics should be used as a kind of personal stepping stone, something to either congratulate themselves for and seek to repeat into the future, or to use as a tool for reflection and motivation for self-improvement. Metrics should not be treated as concrete and immutable, as the majority of metrics collection involves some amount of subjectivity. Hence, an employer

whose employee is underperforming should discuss this with them and suggest areas for improvement to avoid the employee feeling inadequate in the workplace.

5.3 The Consequences of Surveillance

The very process of being ‘measured’ can undoubtedly add an immense amount of pressure to someone’s working life, and in some cases result in hindered performance. An employee may work better when they are not under the judgemental eye of their employer, and as such the mere knowledge that their work is being continuously assessed against numerous metrics could result in less confidence and more stress in their approach to individual pieces of work. Of course, employee progress is frequently measured, as it ought to be, in the modern workplace. However the introduction of more minute and intricate individual assessments could prove dangerous to staff morale. This is especially pressing in software development, where every step of the process has the potential of being monitored.

6. Conclusion

Productivity is an invaluable resource, one which every company hoping to make a profit is seeking to optimise. While conducting this report, it was clear that the perfect measure for it most certainly does not exist. It is exceedingly difficult to decide on which metrics to focus on, let alone the right platform or algorithm through which to do so. Impressive advancements have been made in this regard, but it is obvious that there is much to learn.

With the tools currently at our disposal, however, there is much that can be done, and is being done. By measuring the software engineering process companies have enhanced productivity and decision-making considerably. The field is heading into many exciting new directions, ones which will almost certainly see even more positive results. Yet, we can’t lose our sense of humanity in the pursuit of accurate measurements, as this will cost us greatly.

Bibliography

[1] (Guru99, n.d.)

Accessed at: https://www.guru99.com/what-is-software-engineering.html

[2] (Taylor, 2017)

Accessed at: https://www.klipfolio.com/blog/kpi-metric-measure

[3] (Kimmel, 2016)

Accessed at: https://www.pluralsight.com/blog/teams/5-developer-metrics-every-software-manager-should-care-about

[4] (Ara, et al., 2015)

Accessed at: http://www.hitachi.com/rev/pdf/2015/r2015_08_116.pdf

[5] (Mwila, 2019)

Accessed at: https://hackernoon.com/software-reliability-pt-1-what-is-it-why-developers-should-care-f2a71274772c

[6] (Pan, et al., 2012)

Accessed at:
https://dspace.mit.edu/bitstream/handle/1721.1/92443/Pentland_Modeling%20dynamical.pdf?sequence=1&isAllowed=y

[7] (Passos, et al., 2011)

Accessed at:
http://www.sbgames.org/sbgames2011/proceedings/sbgames/papers/comp/full/30-91552_2.pdf

[8] (Johnson, et al., 2003)

Accessed at: https://ieeexplore-ieee-org.elib.tcd.ie/stamp/stamp.jsp?tp=&arnumber=1201249&tag=1

[9] (GitHub, n.d.)

Accessed at: https://hackystat.github.io/

[10] (Gorski, 2018)

Accessed at: https://www.saasgenius.com/blog/how-find-best-software-measuring-employee-productivity

[11] (GeeksforGeeks, 2020)

Accessed at: https://www.geeksforgeeks.org/software-engineering-halsteads-software-metrics/

[12] (Marshall, n.d.)

Accessed at: https://users.cs.cf.ac.uk/Dave.Marshall/PERL/node32.html

[13] (Britton, 2016)

Accessed at: https://www.perforce.com/blog/qac/what-cyclomatic-complexity

[14] (Guru99, n.d.)

Accessed at: https://www.guru99.com/cyclomatic-complexity.html

[15] (Fenton & Neil, 1999)

Accessed at: https://eecs.qmul.ac.uk/~norman/papers/jss_1999.pdf

[16] (Brownlee, 2019)

Accessed at: https://machinelearningmastery.com/introduction-to-bayesian-belief-networks/

[17] (Sapience, 2020)

Accessed at: https://sapienceanalytics.com/product-overview/faqs/

[18] (Sullivan, 2013)

Accessed at: tlnt.com/how-google-is-using-people-analytics-to-completely-reinvent-hr/

Figure 1: (GeeksforGeeks, 2020)

Accessed at: https://www.geeksforgeeks.org/software-engineering-halsteads-software-metrics/

Figure 2: (IBM, 2020)

Accessed at: https://www.ibm.com/support/knowledgecenter/en/SSSHUF_8.0.2/com.ibm.rational.testrt.studio.doc/topics/csmhalstead.htm

Figure 3: (Guru99, n.d.)

Accessed at: https://www.guru99.com/cyclomatic-complexity.html

Figure 4: (Guru99, n.d.)

Accessed at: https://www.guru99.com/cyclomatic-complexity.html

Figure 5: (Fenton & Neil, 1999)

Accessed at: https://eecs.qmul.ac.uk/~norman/papers/jss_1999.pdf