**Measuring Software Engineering Report**
**Jamison Engels - 17300599**

## Introduction

To understand how the task of software engineering is measured quantitatively, this paper will delve into the following concepts:

- What sort of data can be collected and measured in the software development process?
- What sort of computational systems are in place to automatically collect this data?
- What algorithms are in place that are able to develop insights from this data and make decisions involving productivity?
- What ethical concerns arise from the ways in which software engineers are monitored and evaluated?

## Measurable Data:

As the field of software engineering continues to expand and revolutionize at an ever-increasing rate, the collection of accurate and measurable data has become a vital part of understanding the contributions of each engineer. Lord Kelvin, esteemed Irish-Scottish mathematician, and engineer, famously said, “If you can not measure it, you can not improve it" (Sharlin, 2013). While Lord Kelvin’s words are well over 100 years before the digital age, his assertion on the correlation between measurement and improvement highlights the need for quantifiable data in any profession. However, the metrics used to assess the progress of a software engineer have developed and changed as the career has evolved.

### LOC:

When software engineering began to become a viable profession in the early 1960’s, a very rudimentary method of measuring performance known as LOC or “Lines of Code” was used (Fenton and Neil, 1999). LOC was further expanded to measure the quality of code produced by analyzing the number of defects per one-thousand lines of code (Fenton and Neil, 1999). To the untrained eye, LOC would appear to be the simplest and most straightforward way of evaluating a software engineer as it follows the assumption that writing more code makes a better software engineer. It is easy to see why this method was developed at the inception of software engineering as similar metrics that focus on the quantity of output of a worker are used to assess other professions. However, LOC has obvious and glaring drawbacks that put it at odds with many of the foundational principles of software engineering.

LOC measurements do not correlate well with a software engineer's objective to write clear, efficient, and quality code (Sundar, 2010). When a programmer is judged by their LOC, they are rewarded for writing long and complicated code to artificially inflate their numbers. Furthermore, programmers code in different styles and in different languages, making it hard to compare the output of lower-level languages like C with higher-level languages like Haskell. LOC's greatest shortcoming though comes from its disregard for the requirements gathering, design and testing components of software development and opts to only give recognition to the coding activity (Sundar, 2010). Despite LOC's shortcomings, a 2016 survey of software engineers in Turkey found that around 55% of the participants were still assessed by their LOCs, citing the simplicity of the metric as the primary reason for its continued use (Garousi, 2016). In the absence of LOC, has the industry developed a new catch-all measurement system in the past 50 years? Sadly the answer is no, but new methods to measure productivity and enhance the LOC model have been developed.

**Developed Metrics:**

In a 2018 research paper published in the European Journal of Computer Science titled *Measurable Metrics for Software Process Improvement***,** researcher Supun Dissanayake sought to "identifying measurable metrics that can be used by software development companies to improve the quality of the development process" in the modern world (Dissanayake, 2018). Dissanayake analyzed eleven different methods that are used to measure and gather data about the productivity of software engineers. While not all eleven of these methods will be thoroughly explored, four of them that display the unique metrics that software engineers are judged by will be further developed.

**Reusability Assessment:**

The reusability assessment builds on top of LOC by measuring the reuse code percentage of a project that encourages efficient, maintainable, and readable code (Frakes, 1995). Unlike LOC, a reusability assessment looks for lines of codes that are reused in relation to the total lines of code in the project (Frakes, 1995). By using this measurement, the overall impact of each line of code written by a software engineer can be assessed in its context to the overall system. A software engineer with a high reuse percentage writes elegant and efficient code that can be implemented into other projects and services. However, this metric is not often recommended for object-oriented systems as a different metric would need

to be constructed to understand the reuse of an object rather than an individual line of code (Dissanayake, 2018).

$$Percentage\ Reused\ (\%) = \frac{Lines\ of\ reuse\ code\ in\ the\ system\ module}{Total\ lines\ of\ code\ in\ the\ system\ or\ module} * 100$$

**Work in Progress (WIP):**

Work in progress or WIP is the process of tracking all of the development processes that are in progress at a given time (Little 2008). To track these processes firms use systems like Kanban boards or platforms like Atlassian. The calculate the WIP of a project all unfinished tasks on the respective task tracking system are added up. WIP is a very useful measure to understand how much work is on the back burner for a team and how much work at a time they can handle and manage.

$$WIP = \sum_{i=1}^{n} X_i$$

**Throughput:**

Throughput is the average output of development processes per unit of time (Little, 2008). Different scaling factors such as per day, week, or month can be used to set the time frame of a throughput analysis. Tasks are often known as "tickets" that are given to a team or individual to solve. Throughput can be further broken down and analyzed by focusing on a specific ticket type such as bug fixing or testing to get an understanding of how many of these individual tasks are completed per time frame (Dissanayake, 2018). Throughput is often used to measure the capacity planning to estimate the average workload that can be outputted by a developer.

$$Throughput = \frac{No\ of\ processes\ completed\ per\ day, week\ month\ etc.}{Time\ period\ (day, week, month\ etc.)}$$

**Cycle Time:**

Cycle time is the average amount of time that it takes a development process to be completed (Little 2008). A reduction in cycle time will allow a team to finish tasks at a faster rate and make their deadlines in time (Dissanayake, 2018). Cycle time can be calculated with respect to Little's Law which determines the average number of items in a queue by the

average wait time of each item within the queue (Little 2008). The metrics that were previously discussed, WIP and throughput, are used as the input for the calculation

$$Cycle\ Time = \frac{Work\ in\ Progress\ (WIP)}{Throughput}$$

By looking at this equation, a manager can analyze the productivity of a team in relation to both the tasks they have at hand and the tasks that they are able to complete. To increase throughput, a manager could either try and lower the amount of WIP a team has or can try and encourage them to complete more tasks in a given timeframe. While all of these equations are useful for analyzing productivity and the digital footprint of a developer, how can this data be compiled and structured to make an accurate interference?

**Computational Platforms Available:**

Even though the wide variety of measurable data that can be collected about a developer might make it appear difficult for firms to properly assess a developer's productivity, there are already a wide variety of computational platforms in use by major firms that solve this problem. From planning and organizational platforms like Atlassian to integrated and AI solutions like Velocity and Gitprime, these platforms are in use and will affect developers in the modern age. Some even argue that these platforms are beginning to encroach on the privacy and human rights of developers.

**Atlassian:**

Atlassian is an Australian software company that creates products aimed at "streamlining product development and manage workflow" for tech firms and currently has made over 1.21 billion USD in revenue (Moses, 2010). All 10 of the world's top software companies, such as Microsoft and Oracle, use their products as well as other large global companies such as Shell and Toyota (Moses, 2010). Mark Fidelman, a technology business analyst for Forbes, believes that the quality of product that Atlassian deliver is of such a high level, that firms are naturally attracted without the efforts of sales or marketing, "How many Atlassian salespeople have called to sell you Confluence? You guessed it - zero" (Fidelman, 2013). If one prescribes to the belief that "software is eating the world", then Atlassian can be seen as the head chef that is orchestrating this expansion (Fidelman, 2013). Atlassian offers are variety of platforms that are such as Bitbucket for version control, Bamboo for CI/CD

pipelines, and Trello for collaborative boards. However, the largest and most used platform that Atlassian offers is Jira, an agile development management system.

**Jira:**

Jira Software is an agile project management tool that supports any agile methodology, be it scrum, kanban, or other homebrewed methods (Atlassian, 2019). Jira contains a wide variety of agile reporting metrics that can provide firms deep insight into the performance of a team or developer. In the article *Five Agile Metrics You Won't Hate,* Atlassian developer Dan Radigan digs into some of the ways Jira's metrics can be used to track developers (Radigan, 2019).

Sprint burndown charts track the completion of work throughout the spring by charting the amount of tasks down per day over the period of the project (Radigan, 2019). Jira is able to track this data by the number of completed tasks declared by the developer on their Jira Kanban board. Managers will look at this chart to determine if a project is completed in proper step-wise development within the allotted time frame. Burndown charts can be further expanded for an Epic, or large body of work in Jira, to see the influence of each individual sprint on the project as a whole and check to make sure progress is being made over each sprint (Radigan, 2019). Velocity is the average amount of work a scrum team completes during a sprint and is either measured in story points, development tasks or hours (Radigan, 2019). A project owner or manager usually sets a metric on Jira that the team needs to meet for each spring and then can compare it the actual work completed. Jira's velocity analysis can help understand if developers are able to meet their forecasted goals. Control charts track the amount of time it takes from a story to go from "in progress" to "done" and is a graphical representation of throughput (Radigan, 2019). Control charts are used to guide a team to achieve short cycle time amongst all of their work (Radigan, 2019). Cumulative flow diagrams display how the amount of work to be done grows in relation to the amount of work that is actually completed. Building up WIP, cumulative flow diagrams look for shortages or bottlenecks within work to help smooth the overall development process (Radigan, 2019).

For managers that want more control and data about their developers, Jira supports a wide range of time tracking tools in the Atlassian Marketplace. The "Tempo Timesheets" plugin is one of the most popular add ons for Jira that gives managers in-depth info on each

developer's logged time and workflow. As Jira and its market place continue to expand, Jira will continue to offer more and more ways to evaluate a developer.

**GitPrime and Velocity:**

GitPrime and Velocity are two Engineering Intelligence tools that provide transparency into the work of developers (Rezvina, 2014). Both of these tools are similar in how they combine analytics about the coding habits of developers, but they have a few small key differences that make them unique. Gitlab and Velocity are integrated into git tools such as GitHub or Bitbucket and are linked to specific repositories. Once they are linked, each service will combine different insights into the codebase.

**Velocity:**

Velocity has a very strong focus on pull requests and uses them as the foundation for its data-enrichment algorithms (Rezvina, 2014). Velocity is able to calculate the impact of a change in the code base, the amount of reworking that a pull request needed, and the effort put into each pull request (Rezvina, 2014). Velocity provides high-level tracking of languages and their respective use within a project as well as the code contributions to different areas of a project, such as front-end and back-end. Velocity can break down the contributions of each individual team member through an activity log that visualizes a team member's work cycle (Rezvina, 2014. This data can be used to track each piece of the codebase that the developer interacts with and can easily compare it to the rest of the team through graphical visualizations.

**GitPrime:**

Gitlab is built on git and works primarily off the source-code level data rather than the collaborative pull requests that Velocity works from (Rezvina, 2014). GitPrime is able to measure the same impact and churn data that Velocity is able to but also features data about the speed and quality of code that a developer can produce in both raw and productive settings (Rezvina, 2014). GitPrime is able to dig deep into the number of commits per day and the activity of a developer compared the industry benchmarks. GitPrime provides a similar activity log to Velocity but is able to automatically review each contributor against their team by providing a profile of their work habits and contribution data. This is further

enhanced by the Spot-Check feature that is able to flag anomalies quickly, such as if a developer dropped below their average daily commits. Whether a firm chooses to use GitPrime, Velocity, or other git integrated systems, these computational platforms are able to monitor every piece of code that is added to a repository.

**New Computational Platforms Available**

While services like GitPrime, Velocity, and Jira all focus on tracking the development from the lense of repositories and tickets, new computational platforms are in use that look to gather information about every action a developer takes during their workday. These new platforms can be embedded within the computer itself and flag every non-work-related activity or are even able to track developers' movements throughout the office. Before the ethical concerns of these platforms can be addressed, it is first important to understand how these platforms work.

**Timely**

Timely is an automatic timesheet and productivity tracker that utilizes AI to record work, group it together and build a timesheet of a developer's activities (Memory, 2018). The creators of Timely believe that "millions of us wrestle with digital distraction on a daily basis" and believe that their product can allow people to "redefine [their] priorities" back to their work (Memory, 2018). Timely can be integrated into software such as Intellij, Slack, Chrome, Jira, and Xcode and will create timesheets for developers that will accurately log their productive hours. Timely will automatically not log hours spent on websites like Reddit and Facebook but will include them in a daily breakdown of activities (Memory, 2018). Similar to how GitPrime and Velocity can compare the productivity of a developer to their peers within a git repository, Timely is able to compare developers based on their whole work process.

**Humanyze**

Humanyze is a sociometric solutions company founded by MIT graduates that looks to understand communication and the way work gets down within a firm (Lohr, 2017). Humanzye is giving rise to a new science known as "people analytics" or as the CEO Ben Waber likes to put it, "what goes on internally that people can't answer" (Warber, 2019).

Humanzye collects around 4 gigabytes of data per day about "tone of voice, volume, how quickly you speak, and location within the office" from sensors placed on lanyards around employee's necks (Warber, 2019). From this data, a complex matrix is developed that is able to "predict the performance" of an employee based on all of the collected data or develope insights into how different groups of employees interact with each other (Warber, 2019). A firm that utilized Humanyze found that employees were more productive when they talked with employees from other departments and used Humanyze to change the location of coffee machines every day to get the right departments to meet up (Weller, 2016). Humanyze is also able to identify the connections that employees make with each other and outline naturally forming "cliques" as well as employees that are outside of social circles (Weller, 2016). While Humanyze is a voluntary opt-in system, it highlights an ever-increasing trend of employee surveillance in a potential "Big Brother" like office space. If a firm were to use every computational platform previously mentioned, there would be very few blind spots where an employee can feel like they are not being monitored. With access to such a large amount of data, the next natural step is to apply complex algorithms to it so that automatic inferences can be made. Big Brother soon might not be your boss, but an algorithm.

## Algorithmic Approaches Available

What makes a good programmer? This question has historically been hard to rationalize because "it is difficult to collect useful measures as developers do not consider it an important activity, compared to coding" (Sillitti, et al). Furthermore, the diversification of design, coding, and testing tasks within software development makes it hard to find a "one-size fits all" model that is computable. However, these systems are already being developed and further improved to create algorithmic approaches that are able to gather data and create inferences on what makes a good software engineer.

### PROM

In a 2003 paper published within the Euromicro Conference journal, researches from the University of Genova and Bolzano outline PROM (PRO Metrics), an automated tool for collecting and analyzing software metrics as well as the personal software process (PSP) data (Sillitti, et al). Personal software process (PSP) is a method used to assist software

developers in how to plan and track projects, use a measured and defined process, establish goals, and track their performance against these goals (Bjorn, 2018). To properly utilize PSP, detailed metrics of the development time, bugs discovered and corrected at all development stages, as well as software size are collected (Sillitti, et al). For PROM the application plugins were added popular development environments like eclipse and text editors like emacs to collect and analyze PSP data automatically (Sillitti, et al). Once PROM has collected the data, it analyzes it from the ground up; building from the individuals, up to a company level. PROM aims to analysis the whole development process including activities not strictly related to coding and target users include all members of the development team (Sillitti, et al) PROM uses an algorithm that relates the effort that was put by the user into writing code and then determines the significance of their code to generate a "total effort" figure. From this algorithmic approach and data collected, PROM can help managers improve both the code and process improvements of their developers automatically (Sillitti, et al).

**CoSEEEK**

Context-aware Software Engineering Environment Event-driven frameworK (CoSEEEK) was a process developed by researchers at Allen University to "improve SE process support and guidance in an automated fashion" (Grambow, et al, 2013). CoSEEEK aims to "automatically gathering information from the environment and users, uniting it with information from a knowledge base, and utilizing this information for on-the-fly process optimization" that can automatically manage teams of software engineers (Grambow, et al, 2013). To do this, CoSEEEK utilizes automated detection of quality problems in the source code by assigning quality measures to code (Grambow, et al, 2013). CoSEEEK uses a complex algorithm to create an *Assessment Process,* rate it from different *Base Practices* and calculates its achievement of capability (Grambow, et al, 2013).

While a full in-depth breakdown of the algorithm can be found within the paper, the algorithm that CoSEEK uses can be broken down into three steps. First, an *Assessment Process Template* is created with and linked to *Process Categories, Processes,* and *Base Practices,* to be used as metrics (lines 01-14). Then *Capability Levels* and there attached *Determinators* are linked to *Base Practices* and assigned a *Performance Scale* for the respective employee using ISO/IEC 15504 (SPICE) standards for completeness (line 15-27). This step in the process correlating people and their effort to a task in the *Assessment*

*Process.* The final part of the algorithm compiles all of the ratings at the end of the project (or anytime an assessment is desired) and deliveries to the manager for review and allows them to fill in any missing information. The following pseudo-code was given to display the functionality of this system:

```
AssessmentProcess AP ← createConcept(APT)
linkConcepts(P, AP)
for all APT.processCategoryTemplates PCT do
    ProcessCategory PC ← createConcept(PCT)
    linkConcepts(AP, PC)
    for all PCT.processTemplates PT do
        Process PR ← createConcept(PRT)
        linkConcepts(PC, PR)
        for all PRT.basePracticesTemplates BPT do
            BasePractice BP ← createConcept(BPT)
            linkConcepts(PR, BP)
        end for
    end for
end for
for all APT.capabilityLevelTemplates CLT do
    CapabilityLevel CL ← createConcept(CLT)
    linkConcepts(AP, CL)
    linkConcepts(CL, CLT.calculatedFor)
    for all CLT.capabilityDeterminatorTemplates
            CDT do
        CapabilityDeterminator CD ← createConcept(CDT)
        linkConcepts(CL, CD)
        List relatedBPs ← getRelatedBasePracts(CD, AP)
        for all relatedBPs BP do
            new rating(CD, BP,
                AP.getStandardPerformanceScale,Pers)
        end for
    end for
end for
automatedRating(AP)
manualRating(AP)
for all AP.capabilityLevels CL do
    checkAchievement(CL)
end for
```

While the complexity of CoSEEK and the researchers work is beyond the scope of myself and many others comprehension, the takeaway from the CoSEEK algorithm is that algorithmic systems are in use that are able to assess an employees work automatically by assigning ratings to individual tasks and work that are then compiled into a complete report for managerial review.

**Network Model for Worker Productivity**

To understand how co-worker productivity affects worker productivity via network effects, researchers from Stockholm University collected data from a large multi-national network operator and modeled these effects through an algorithmic approach (Lindquist, et al 2015). Researchers investigated how on-the-job training of workers affected the overall productivity of the office, even the productivity of workers who were not trained. The driving question of their research was boiled down to “does co-worker productivity affect worker

productivity; if so, is it possible to identify key workers in the firm?" (Lindquist, et al 2015). Over a period of two years, data was collected from a multi-national mobile network operator which included information such as, detailed information on the performance of individual workers, their characteristics, their team affiliation, and the exact times that they punch in and out of work (Lindquist, et al 2015). This data allowed researchers to construct co-worker networks that "weighs" the time co-workers spent together, creating associations to see how productivity spills over. Researchers found that a 10% increase in the productivity of a worker's network resulted in a 1.7% increase in a worker's own productivity. To come to this conclusion a Nash Equilibrium was used as the base for their calculations. Nash Equilibrium is a concept in game theory "where the optimal outcome of a game is where there is no incentive to deviate from their initial strategy" (Chen, 2019). For this paper the following Nash Equilibrium was used:

$$y_i = \phi_1 \sum_{j=1}^{n} g_{ij} y_j + \phi_2 \sum_{j=1}^{n} g_{ij}^{*} y_j + \alpha_i$$

In a system where workers choose their effort level, each worker will maximize their utility in relation to the aggregate and average input of their peers in an effort to conform to their work standards (Lindquist, et al 2015). While the full explanation for this equation can be found within the paper, in simple terms, this equation states that works will match their input to that of their peers so that neither party has an advantage over each other. The second concept that the researchers looked into was the "key players" within an organization. A key player is defined as "the agent that should be targeted by the planner so that, once removed, she will generate the highest level of reduction in total activity" (Lindquist, et al 2015). Through the use of an adjacency matrix where the aggregate effort level of an individual their corresponding network is inputted, the following complex equation was used to find the key players:

$$d_i(g, \phi_1, \phi_2, \boldsymbol{\alpha}) = B(g, \phi_1, \phi_2, \boldsymbol{\alpha}) - B(g^{[-i]}, \phi_1, \phi_2, \boldsymbol{\alpha}^{[-i]}).$$

Similar to the previous equation, I was not able to grasp the full complexity of how this information is calculated, but this equation, researchers were able to find the key players within a firm as well as measure the impact of the removal of a key player. These insights and further analyze allowed the researchers to come to the following conclusions:

(i) The firm should increase team size

(ii) The firm should hire more "bridge" workers to spill over knowledge between teams

(iii) The firm should increase the average connections in each worker's network

From this paper, it can be seen how a deep and complex analysis can be given to understand how important a worker is, how much they are contributing to their peers, and how they exchange knowledge with each other. Similar to how Google is able to analyze a webpage to determine it's page rank, all of the algorithms that were explored in this section show how the "importance" of a worker can be boiled down into a calculable figure given info data. This data could be automatically used by a higher-level system to make hiring decisions, fire employees, or even micromanage every aspect of a firm. However, is this a future we as a society want to have?

## Ethics Concerns - My Own Opinions

Before I dive into my own personal "gut reaction" to researching and writing this paper, I want to introduce a philosophical concept I learned about a few years ago that I find eerily relevant to the topic of this paper.

### The Panopticon

The panopticon is a disciplinary concept introduced by the English philosopher Jeremy Bentham in the 18th century (Ethics Centre, 2017). Bentham believed "that power should be visible and unverifiable" and is manifested through the layout of an "ideal" prison (Ethics Centre, 2017). The prison is designed so that there is a central observation tower placed within a circle of prison cells. From the tower, a guard can see every cell and inmate, but the inmates can't see into the tower, leading to a prison where the prisoners never know whether or not they are being watched by the guards (Ethics Centre, 2017). Bentham theorizes that systems that put individuals under constant surveillance would be able to bend and shape society, "morals would be reformed, health preserved, industry invigorated"

(Ethics Centre, 2017). Today, many philosophers argue that computers are in some way a modern panopticon. When an average user logs into a computer, they are often unaware of the full expanse of the data that is being collected about them and how that data is being processed. Are we all simply digital prisons?

**The Panopticon in the Workplace**

As previously explored in this research paper, there are numerous systems and algorithms that can monitor a software engineer in their workplace. All of these systems are easily integrated into any workplace and are able to capture almost every bit of data a manager wants. Even though many of these systems have "opt-out" functions, as displayed in the call center study, employees will naturally follow along with the actions of their pears if it is perceived to be an increase in their own productivity. Once these systems are in place, they appear to be naturally expanding and consuming. Furthermore, even engineers with a strong technical background may not be able to grasp the whole complexity of what data is collected and how they are being assessed. Under the panopticon in the workplace, workers would never know if their one-time extra 10-minute long lunch break is being factored into a complex algorithm that determines their pay. This could lead to an environment where employees fear the systems that were put in place to make their lives better and reduce stress in the workplace. While this might appear to be an over-exaggeration, the EU has already responded with these rising concerns by implementing a "right to explanation" that states that users of a system are entitled to ask for an explanation of how an algorithm makes decisions (Ethics Centre, 2017). The fact that the EU has already come forward with a possible solution to this problem justifies employees' concerns who are not protected by it. From an ethical standpoint, the long term constant surveillance of employees appears to force employees to give up their privacy and well being in the name of productivity.

**Accuracy of Data**

A common concern that software engineers have regarding their data is its accuracy. As previously explored, evaluating software engineering is difficult. Every different metric rewards some way of coding while discourages others. If I knew that LOC was the primary metric that a firm uses, I would most certainly write longer less efficient code if I knew I was rewarded for it. On the other hand, if compact and efficient code was how the firm judged my

work, I would constantly be spending time trying to min-max each function, which could possibly take away from my time on other work. Furthermore, what if there is a bug in the code, what if the data that is being taken is not accurate? If some part of the algorithm or system does not function as intended, could I get demoted, or even lose my job? These questions surely fill the head of any software engineer as having you productivity judges by a system that you are all too familiar with breaking is a scary thought. In places without proper GDPR protection, this data even has the potential to be sold off or misused if not protected. There have been many large scale data leaks from in recent history and the vast amount of information that could be collected about an individual as the go-to work every day is ripe for the picking. The sheer amount of data that is being collected and stored every day could lead to problems managing it and correcting it. What sort of employees would be responsible for this? I worry that all of these concerns would not be dealt with both responsible and ethically by the many firms that would employ these systems.

**The Future of the Profession**

Because software engineers work in systems that are very easy to monitor, I worry that the future of the profession will result in an exponential arms race to improve productivity. Some employees might love these systems as they are able to show more commitment to their job and are rewarded for min-maxing their work life. These employees might love the fact that a system is able to recognize that they have been taking a shorter lunch break to get more work done. I worry however that this could result in a hostile work environment where employees look to find every little advantage they can have within the algorithm. These hyper-competitive employees will most likely raise the overall productivity of their peers but would force others to keep up or get left behind. It could possibly even force employees to make decisions that hard their own mental and physical well being simply on the basis that “others are doing it to”. I believe that this would not result in the success and happiness of many employees that would otherwise be equipped to handle a job in the industry.

**Silver Linings?**

Even though I have highlighted many of my concerns with the way software engineers are tracked, there are a few good silver linings. If these systems are put to good use

and managers actually care about their employees, it can certainly be used to fix many problems within the workplace. Projects and groups could easily be formed based on measurable skills and time management so that teams are optimized to complete their work on time with equal effort. Offices could be redesigned and modified to encourage more social and cooperative teams. Algorithms and AI could possibly be used to prevent workplace discrimination by removing racial, gender, or sexual components from the evaluation process. While all of these are possible I fear that these systems will ultimately be misused in the long term. These same systems could actually make it worse for minorities as pre-built bias may arise from those who developed them that can be shrouded in mathematical complexity. When push comes to shove and profit is put before people, the systems that are marketed as forces of good can easily be used to further the divide between employee and employer.

## Conclusions

There is a famous quote from Richard Feynman that I would like to conclude this paper with:

"To every man is given the key to the gates of heaven. The same key opens the gates of hell.

And so it is with science"

The tools that we as software developers are tasked to build every day have equal potential to do good as they do bad. At the end of the day, it is up to use them properly. We need to build a just society that is able to make sure these tools carry each other up rather than tear each other down.

# References


Atlassian. (2019). *Agile tools for software teams - Jira Software | Atlassian*. [online] Available at: https://www.atlassian.com/software/jira/agile [Accessed 26 Oct. 2019].

Bjorn, W. (2018). *What is Personal Software Process? - PSP*. [online] Explainagile.com. Available at: https://explainagile.com/agile/personal-software-process/ [Accessed 5 Nov. 2019].

Chen, J. (2019). *Nash Equilibrium*. [online] Investopedia. Available at: https://www.investopedia.com/terms/n/nash-equilibrium.asp [Accessed 7 Nov. 2019].

Dissanayake, S. (2018). Measurable Metrics for Software Process Improvement. *European Journal of Computer Science and Information Technology*, 6(1), pp.33 - 43.

Fenton, N. and Neil, M. (1999). Software metrics: successes, failures and new directions. *Journal of Systems and Software*, 47(2-3), pp.149-157.

Fidelman, M. (2012). *Why Atlassian is to Software as Apple is to Design*. [online] Forbes.com. Available at: https://www.forbes.com/sites/markfidelman/2012/05/04/why-atlassian-is-to-software-as-apple-is-to-design/#35b261764376 [Accessed 26 Oct. 2019].

Frakes, W. (1995). *Software reuse. In Encyclopedia of Microcomputers*. New York: Marcel Dekker Inc, pp.179-184.

G, G., R, O. and M, R. (2013). Automated Software Engineering Process Assessment: Supporting Diverse Models using an Ontology. *Journal on Advances in Software*, 6, pp.213-224.

Garousi, V., Coşkunçay, A., Demirörs, O. and Yazici, A. (2016). Cross-factor analysis of software engineering practices versus practitioner demographics: An exploratory study in Turkey. *Journal of Systems and Software*, 111, pp.49-73.

Grambow, G., Oberhauser, R. and Reicher, M. (2013). Automated Software Engineering Process Assessment: Supporting Diverse Models using an Ontology. *Complex Networks*, pp.pp. 187-198.

Little, J. (2008). Little's law. Building Intuition. *Massachusetts Institute of Technology*, Chapter 5, pp.81-100.

Lohr, S. (2017). *Don't Get Too Comfortable at That Desk*. [online] Nytimes.com. Available at:

https://www.nytimes.com/2017/10/06/business/the-office-gets-remade-again.html [Accessed 27 Oct. 2019].

Memory.ai. (2019). *Memory Manifesto*. [online] Available at: https://memory.ai/manifesto/ [Accessed 27 Oct. 2019].

Moses, A. (2010). *From Uni dropouts to software magnates*. [online] The Sydney Morning Herald. Available at: https://www.smh.com.au/business/small-business/from-uni-dropouts-to-software-magnates-20100715-10bsm.html [Accessed 26 Oct. 2019].

Oliveira, E., Viana, D., Cristo, M. and Conte, T. (2017). How have Software Engineering Researchers been Measuring Software Productivity? - A Systematic Mapping Study. *Proceedings of the 19th International Conference on Enterprise Information Systems*.

Radigan, D. (2019). *Five agile metrics you won't hate | Atlassian*. [online] Atlassian. Available at: https://www.atlassian.com/agile/project-management/metrics [Accessed 26 Oct. 2019].

Rezvina, S. (2014). *Velocity vs. GitPrime: Choosing an Engineering Intelligence Tool*. [online] Code Climate. Available at: https://codeclimate.com/blog/velocity-vs-gitprime/ [Accessed 26 Oct. 2019].

Sharlin, H. (2013). *William Thomson, Baron Kelvin | Biography & Facts*. [online] Encyclopedia Britannica. Available at: https://www.britannica.com/biography/William-Thomson-Baron-Kelvin [Accessed 21 Oct. 2019].

Sillitti, A., Janes, A., Succi, G. and Vernazza, T. (2003). Collecting, Integrating and Analyzing Software Metrics and Personal Software Process Data. *Euromicro Conference*, 29, pp.336 – 342.

Sundar, D. (2010). *Software engineering*. New Delhi: University Science Press.

THE ETHICS CENTRE. (2017). *Ethics Explainer: The Panopticon - What is the panopticon effect?*. [online] Available at: https://ethics.org.au/ethics-explainer-panopticon-what-is-the-panopticon-effect/ [Accessed 7 Nov. 2019].

Waber, B. (2019). *Technology for workplaces that work: Humanyze's Ben Waber*. [online] MIT Technology Review. Available at: https://www.technologyreview.com/s/612814/technology-for-workplaces-that-work-humanyzes-ben-waber/ [Accessed 27 Oct. 2019].

Weller, C. (2016). *Employees at a dozen Fortune 500 companies wear digital badges that watch and listen to their every move*. [online] Business Insider. Available at:

https://www.businessinsider.com/humanyze-badges-watch-and-listen-employees-2016-10?r=US&IR=T [Accessed 27 Oct. 2019].