Introduction

The matter of whether, and how, a software engineer's productivity can be measured is frequently a topic of debate and of research. The same issue stands for almost every kind of profession that deals with the large teams and abstract thinking that computer science does; what it the right way to do it, if there even is one?

There have been various attempts at producing an answer, with varying degrees of success. The primary concern is that the data needs to be understandable to a manager, or some other such person who may not have experience with programming. However, the level of detail, report writing, and analysis needed to achieve this runs the very high risk of decreasing productivity itself by forcing engineers to interrupt their work frequently to fill out spreadsheets or take measurements in some other such way.

So, while I will be exploring the various ways the different elements of computer science work can be measured, it's important to consider that this is a topic of high contention and that each solution has its strengths and flaws.

<u>Measurable Data</u>

To begin, we will explore Measurable Data. That is, hard data that can be measured from the software engineering process and what can be derived from it.

Problematic Simplistic Methods of Measuring Data

Measuring a developer's productivity is difficult in two significant ways; one, the method of measuring becomes so extensive and detailed that it can disrupt workflow and absorb time, and two, the metric being measured is not relevant to productivity and, in fact, may harm it. Frequently, the most simplistic attempts at measuring productivity fall in the latter category. Perhaps one may try to measure productivity by counting the lines of code (LOC or KLOC for thousands of lines of code (Fenton, N. E., and Martin, N. (1999)) produced by a developer or team. The issue with this method is that it rewards inefficiency, where good code is concise and is kept as simple as possible. (Hakes, 2018) Another method may be to measure hours worked but the same problem as above arises (Hakes, 2018), especially as there is significant proof that working more hours a week reduces overall productivity, and not just in software engineering (Cs.stanford.edu, 2018).

Measuring the number of bugs fixed or tasks completed may also be an option, but similar problems arise. Tasks may take equally long to complete but may be more or less important than each other. While rewarding bug fixes promotes poor quality initial code.

Important Metrics for Measuring Data

Rather than attempt to measure productivity by the above listed, it is frequently measured in ways that consider the quality of the code and the developer's speed and efficiency. The quality of the code may refer to how many requirements it fulfils, how many of the needed functions or structures it contains and how efficiently they were implemented (Kanat-Alexander, 2018).

Of course, these metrics are much harder to measure and are usually defined by the manager of the team or developer being measured. For example, depending on the quality required, the code may need to be extensively tested or fast or memory efficient.

Senior Software Engineer Nick Hodges suggests that although these metrics are subjective, they can still be measured. His suggestion is to measure productivity by looking at a

developer's engagement in the project, meaning their commitment to it and to producing good work, and also to whether the developer is adhering to coding standards, guidelines and project requirements. He suggests paying particular attention to the quality of the work produced; whether the developer's code works as designed and is thoroughly tested and relatively bug-free (Hodges, 2012).
In the following sections, we will explore methods that take these metrics into account with varying degrees of specificity and success.

Alternative Options - PSP and Leap

A frequently used tool for not just measuring data but also planning projects is the Personal Software Process, or PSP. The intention of the PSP is to aid developers to improve their productivity by carefully planning and structuring a project and by comparing their plans versus the actual result of their planning during the making of code.
The general structure of the PSP is to begin with planning the project. Then the design for the project is written out and reviewed. Then the code is written, reviewed, compiled and tested. During each step, the engineer is following a PSP script which guides them through the process. Along with this, they are recording their time and any defect data, i.e., problems in the work process that can be identified to prevent from occurring later on (S. Humphrey, 2000).
The final step in the PSP is to perform a "post-mortem", an analysis of the engineer's expected productivity versus their actual. This step is intended to improve the engineer's work in the future by identifying errors and setbacks that may occur be preventable (S. Humphrey, 2000).
However, as mentioned before, the process of filing in reports, preparing lengthy plans and reflecting on previous projects is a significant time sink, one that may worsen the work process.
The Collaborative Software Development Laboratory (CSDL) at the University of Hawaii at Manoa have been working to improve analytics for developers and improve the development process. They began research by using a model of the PSP which utilised "simple spreadsheets, manual data collection, and manual analysis" (M. Johnson, 2013). While this model was accurate and powerful in its flexibility, it was extremely clunky and time consuming to use. It required developers fill out a large variety of forms including time-recording logs, size and time estimation templates and defect-recording logs. According to the Philip M. Johnson, director of the CSDL, the "forms typically yield more than 500 distinct values that developers must manually calculate" (M. Johnson, 2013).
As is such, the manual nature of the PSP led to frequent errors in calculation and "incorrect process conclusions" (M. Johnson, 2013), that is, the wrong conclusions were derived in the post-mortem phase about how to improve the development process.
An attempt to fix this was to develop the LEAP (lightweight, empirical, antimeasurement dysfunction, and portable software process measurement) toolkit. The toolkit still needed a developer to manually enter most data, but the LEAP could provide analysis for the data provided. However, the LEAP was not an easy solution. In Johnson's words, "By introducing automation, the Leap toolkit makes certain analytics easy to collect but others increasingly difficult" (M. Johnson, 2013). Now, if a developer wanted to analyse a different aspect of their work process that they thought might be affecting their productivity, they would need to design a new LEAP tool that could analyse for them.

The CSDL would continue with their research, designing other tools like Hackystat in efforts to further improve the measurement process, but this is where their exploration of the PSP and LEAP stops. They do, however, leave a very poignant observation, “The easier an analytic is to collect and the less controversial it is to use, the more limited its usefulness and generality” (M. Johnson, 2013).

TSP

The TSP, or Team Software Process, refers to the guidelines designed to help teams of developers, usually between the size of 2 to 20, to measure and improve their productivity. The TSP works hand in hand with the PSP, any team which implements the TSP will most likely expect each individual to self-evaluate using the PSP (S. Humphrey, 2000).
A team must be defined here as two or more people working on a shared goal. Any team large enough to work on separate sub-tasks but be evaluated by a larger-scale TSP, or CMM (Capability Maturity Model) (S. Humphrey, 2000).
The TSP begins by establishing the goals of the project and assigning tasks, if necessary at this stage, the project may be divided into smaller teams with specific tasks. Once this has been established, the TSP provides guidelines for how to produce a strategy for project development, testing, and risk assessment (S. Humphrey, 2000).
Similar to the PSP, it contains a post-mortem phase which requires that the team analyse its expected and actual productivity and identify any places where errors were made which can be improved upon in later projects (S. Humphrey, 2000).
The TSP is a natural extension of the PSP and needs careful consideration as a form of measuring the development process as, while cumbersome, it is highly accurate.

Computational Platforms

Next, we explore Computational Platforms. That is, the tools used to measure any metrics related to developer productivity. Already mentioned above are tools like the PSP and TSP. Although apart from using the accompanying LEAP, conclusions cannot always be easily drawn about the developer's productivity, even if solid metrics have been measured. To that extent, various methods and tools exist to record and measure the work process.

Social Tools

Social tools, applications that allow employees to communicate among each other or with clients, are being used more and more. This is especially apparent in the field of software engineering where teams may not work in the same location and may need to keep in touch constantly through various tools. These include online video conferencing like Skype, social networking, collaborative document editing and video sharing (Bughin and Chui, 2013).
There is evidence to show that the number of companies that use at least one social tool or technology is climbing (Bughin and Chui, 2013), so without a doubt this will become a vital part of a software developer's work.
Research by Jacques Bughin and Michael Chui, partners and researchers at McKinsey and Company, shows that the use of social tools yields some significant and measurable benefits. For a developer, who may not be in contact with external forces like customers very often will experience benefits like increased speed of access to knowledge and internal experts, communication and travel costs reduced, and increasing employee satisfaction (Bughin and Chui, 2013).
The advantage of social tools, for the purposes of this report, is that it can all be measured. Regardless of what platform a tool is used on, the developer's use of it can be measured and recorded. One can measure the time they spend in calls, the emails and the quality of the information they exchange with their co-workers, the amount of data exchanged within the team and what it is about, and even what the developer uses their devices for and how much time they spend on applications.
This can all be used to track and measure the amount and quality of a software developer's work.

Measuring Mood

A developer's mood can also be a useful indicator of work ethic and can affect productivity. As researchers at Hitachi Limited have shown (Yano et al., 2015), compared to people who are unhappy, people who are happy have 37% higher work productivity. However, as the study points out, quantifying happiness is not accurate, if at all possible.
Their approach was to use questionnaires but then they moved into using devices to measure employees' physical metrics. After enough research, they were able to find a strong correlation between a person's physical activity and their overall mood. They did this by designing a device for subjects of the study to wear and then giving them questionnaires. Soon, they found what they refer to as the "1/T rule" (Yano et al., 2015).
The rule proceeds as follows "It categorizes physical activity during each unit of time as either inactive or active, and looks at the "active" times when the person is moving and how long they last" (Yano et al., 2015), the duration of activity, in this case, is referred to as T.

The study reveals that it was discovered that this 1/T value “is strongly correlated with happiness” (Yano et al., 2015).
The range of the study is quite large, they attached sensors to 468 employees from 10 departments across seven companies to collect approximately 5,000 days of data (Yano et al., 2015), and then the physical data was compared to the results of a questionnaire meant to measure happiness, a very strong correlation was found between the two (Yano et al., 2015).
The study concludes that the more physically active an employee was, the happier, and thus, more productive they were (Yano et al., 2015). The result of this is hugely useful to other companies. For one, there is now a way to record this metric on employees to measure their happiness and thus their productivity. For another, this shows that a developer’s productivity could potentially be raised by implementing changes in the workplace which encourage greater physical activity, among other things.

The Seven Basic Tools of Quality Measurement
Finally, we arrive at the seven basic tools of quality measurement, designed by Kaoru Ishikawa, professor at the Faculty of Engineering at the University of Tokyo. The seven basic tools are widely used and are simple in contrast to tools like the PSP and are explained here with the aid of Stephen H. Kan’s book on the topic.
The first of these tools is a check sheet. Its main purpose is to gather and organise data while also checking off and recording completed tasks. Kan points out that the checklist plays “a significant role in software development” and explains that, in his past experience with work, large projects would frequently be divided up into smaller parts, all with various tasks associated with them and several project phases. In this environment, which is common in software development, checklists are hugely useful (Stephen H. Kan. 2002). Checklists include those which measure tasks to be done and errors found in the software and can be kept manually or through automatic means.
The next is a Pareto diagram, which is a frequency chart of bars, representing the number of errors found, in descending order. It’s used to identify parts of the project or software which cause the most problems. These problems can take a variety of forms, either it’s a piece of code that a large number of bugs seem to originate from or it’s an aspect of the project that drains time and resources (Stephen H. Kan. 2002).
The purpose of this graph is to identify these problems and, primarily, identify which aspects of the project are causing the most problems. Once identified, it can be altered or fixed to improve the productivity of later work.
Next is a histogram, which is similar to the Pareto diagram in purpose. It also contains the frequency of errors in a project except that the bars or organised by the severity of software defects, rather than by order of size. The purpose of the histogram is to enhance understanding of the areas of the project that are problematic, i.e. problem areas may cause less frequent errors, but the severity of the errors results in significant cause for concern (Stephen H. Kan. 2002).
Fourth are run charts, which serve to record productivity by various metrics (e.g. errors in code, quality of code, amount of code written) over time, usually over the course of the project or in work weeks and months. These charts can then be compared to previous projects, or against a projection model, in order to ensure the project is on track and improving upon previous assignments.

As Kan points out, “the goal is to ensure timely deliveries of fixes to customers” (Stephen H. Kan. 2002).
The next is a scatter diagram, which is a more difficult tool to apply, Kan states, as it requires precise data. The graph can be used in various ways, it can show program complexity against defect level, the relationships among defects, relationship between testing defect rates and so on (Stephen H. Kan. 2002).
This graph can be used along-side the above graphs to help identify errors, severity of errors and point of origin, along with providing an excellent metric to track project progress against previous assignments.
The next tool is the Control chart, which is “an advanced form of a run chart for situations where the process capability can be defined” (Stephen H. Kan. 2002). It consists of a central line buffered by upper and lower limits, plotted on it are values of the project that are considered important, for example the time taken to complete a certain task. If the plot remains within the two limits, then there is not cause for concern. However, if it dips above the maximum or below the minimum, then there exists the distinct possibility that a developer or team is working incorrectly, either very inefficiently or they may not be giving a task enough attention.
Finally, we have the cause-and-effect diagram, developed by Ishikawa himself. The diagram plots phases of the project along with errors or characteristics. The characteristics then point to other elements of the project that they will directly cause (therefore, a cause-and-effect diagram). This diagram, while less widely used, is useful in quality improvement and measuring the steps and tasks taken by a development team.
The above tools can be used in a variety of ways to achieve different measurements, but they are without a doubt widely used and useful (Stephen H. Kan. 2002). With spreadsheets and graphs, tracking the progress of a project can be simplified, along with giving metrics with which a developer’s work ethic and productivity can be measured.

Algorithms and Approaches

In the above are discussed the potential metrics that can be measured to track a developer's work ethic, along with the tools that can be used in this process. Algorithms and approaches refers to the potential algorithms that can be used to measure a developer's progress through a project; the various methods for collecting data and understanding which data is worth collecting. It also refers to the algorithms used once the data is collected to derive information from it.

The most popular form of algorithm will attempt to make some connection between quantity, that is, metrics like lines of code written or commits made, and quality, meaning metrics like code coverage, efficiency and tests passed. Both the methods used to collect and visualise data will commonly attempt to reconcile the two, though with various results. Firstly, we will discuss the algorithms used above.

Previously Mentioned Algorithms

Mentioned previously were tools which utilised various algorithms to measure metrics. The PSP and TSP were tools used to record and report but did not provide algorithms to either collect or analyse data. However, software like the LEAP was able to interpret the data given to it by a developer using various algorithms to aid in analysis. This prevents, or lessens, the issue of disrupting a software developer's work and prevents errors in misinterpreting the results.

Then we have the mood measuring method of quantifying developer productivity. This method comes with a very specific algorithm for interpreting the found data, named the "1/T" rule by the researchers who discovered it, which finds a strong correlation between how much physical activity a subject partakes in during work hours and how highly they rank their mood on a questionnaire.

Finally, we have the seven tools of quality measurement, described in detail above. In short, these graphs combine various measurements to provide users with very specific details about their projects, the amount of errors they produce, the severity of the errors, the focal point of the errors, time taken to complete certain tasks, etc.

Collecting Data

As mentioned previously, collecting simple metrics like lines written or number of bugs doesn't often give a clear impression. Automated tests and commit frequency may be in the right vein but don't guarantee code quality. They can, however, be measured in combination.

One method is to divide improvements to the code into their minimum unit of meaningful work (Dario, 2018). These improvements may include new features, enhancements and bug fixes, and must add value to the current product. This is the measurement of quantity.

As each new piece of code must be tested and reviewed. Each successful review of the code can be used as the minimum unit of quality, or meaningful work (Dario, 2018). Each review, when paired with its respective minimum code, gives more dimensions to the new work done; whether the developer created bugs, whether code coverage has increased or decreased and whether tests are failing and why.

Another method is one developed and researched by the Collaborative Software Development Laboratory (CSDL) at the university of Hawaii at Manoa, who had been previously researching the PSP and LEAP and their uses. As previously mentioned, they developed Hackystat, a framework for collection, analysis and visualization of data.

Hackystat is able to collect a software developer's data in the background while they work, and therefore not interrupt them like previous tools (M. Johnson, 2013).
It collects data about the developer's works, like when they started work and everything they've been doing in 5-minute intervals. It measures which files were edited the most often and at what time of day and how many and which tests were invoked and whether they passed or failed, along with how many times a developer has committed their work to a repository and how many times they've built their system and whether those builds were successful or not (M. Johnson, 2013).
This information reveals much about the quantity and quality of a software developer's work, similar to the above-mentioned method. However, once this information has been collected, interpretation and visualisation must be considered.

Data Interpretation Algorithms
An algorithm that can be used to interpret activity data is outlined in Jean Heile, Ian Wright, and Albert Ziegler's paper on using machine learning to measure software development productivity (Heile, Wright and Ziegler, n.d.). Their approach is to use a Hidden Markov model (HMM), a statistical model in which the system being modelled is assumed to have hidden states to be interpreted, to predict the probability that a developer is coding and thus quantify their productivity.
The model is given access to a developer's commitments to a repository including their times. The algorithm assumes that, during the time between two commits, the developer is writing the code for the second commitment, skewed to be at their most productive after the committing of the first piece of code and right before the committing of the second piece.
Along with this, the paper describes an algorithm and dataset called LGTM ('looks good to me') which contains "over 10M commits submitted by ≈ 300K developers to ≈ 56K open source projects" (Heile, Wright and Ziegler, n.d.). LGTM executes hundreds of queries on the supplied code to attempt to identify potential problems, from simple syntax mistakes to more complex ones. With the LGTM and HMM, we have two measures of quantity and quality.
In the next section, we will explore how these values can be visualised.

Visualising Analysed Data
Without graphs to visualise the analysed data, meaningful interpretation would be extremely difficult. Luckily, many analysis algorithms produce metrics that can be easily represented.
The paper which discusses the HMM and LGTM also provides some graphs to visualise the found data. A graph of the potential results given by the HMM can be seen in Figure 2 of the referenced paper (Heile, Wright and Ziegler, n.d.), which presents the likelihood of code being written between commitments along with an estimation of how much time is spent coding and non-coding. Figure 4 (Heile, Wright and Ziegler, n.d.), shows us a code quality comparison chart using LGTM. The quality score is found using LGTM's hundreds of queries and is represented in the y-axis as a value between 0 and 1 against the lines of code for several displayed projects.
The former graph grants the ability to track a software developer's productivity in terms of how often they are writing code between commits. While the latter shows multiple projects

from multiple developers, allowing for comparisons of productivity or activity between teams.
The paper by the CSDL mentioned above already shows some methods for interpreting Hackystat's data. Figure 3 in the paper (M. Johnson, 2013) shows a table displaying the values collected including development time, number of commitments, number of builds and various test data for different projects. This allows comparisons between projects but also between developers if the need arises. Another alternative, however, is a run chart showing the various peaks and troughs of developer productivity, somewhat similar to the previous graph displaying HMM results.
Along with these, various algorithms exist which analyse projects in terms of how much a developer has contributed to the project and in what areas. These values can be represented in the form of connected items, usually circles with varying size depending on lines of code or relevance of code (Dario, 2018). The items are connected in terms of whether they reference each other and can be coloured differently depending on topic, language or the developer who submitted it. This allows for easy comparison between developers, for example, in the provided reference (Dario, 2018), we see that the project leader is responsible for much more code than more junior members. However, if we scale the items of code based on quality rather than size, we'll be able to see which members make the most meaningful contributions. In this way, we can compare various metrics between developers and projects.

Ethics

Here, Ethics is in reference to two separate things; the ethics of measuring a developer's work process and how ethics affects a worker's productivity, that is, how their work and environment affects their performance.

Ethics Behind Measurement

Carefully micromanaging and recording a developer's work can be somewhat controversial. It's rare to find someone who is comfortable having their work meticulously watched and recorded, even if it is for the betterment of the project. Along with that, frequently the tools used to collect these metrics are problematic, as they may record sensitive information or may not be fully protected.

As of current, in Ireland, workplace surveillance is legal and generally thought to be ethical, on the condition that the reasons for doing are legitimate and necessary, as they would be for the purposes of monitoring a project (Citizensinformation.ie, n.d.).

For one example, we have social tools, as previously mentioned, which are useful modes of communication and information exchange. However, these tools carry sensitive information which needs to be protected carefully. According to research by Bughin and Chui, using social tools at work comes with several risks. These include the increased likelihood that confidential information can be leaked, the increased risk of "inappropriate intellectual property distribution", the possibility of distracting employees from core tasks (which is a risk posed by the majority of measurement tools, as mentioned above, the PSP especially is distracting and time consuming to fulfil), and inappropriate behaviour on the part of the employee, which can reflect negatively on the project or company (Bughin and Chui, 2013).

Ethics and Software Development

As software frequently stores sensitive information about an individual; their name, address or bank account details, to give a few, it is the responsibility of the developer to consider the ethical repercussions. Information can be collected from various sources, in a process called data fusion, to be used unethically (J. Thomson and L. Schmoldt, 2001). Or perhaps a sensitive location is stored which, when exposed, can leave individuals or groups of people exposed to various dangers (J. Thomson and L. Schmoldt, 2001). This leaves software development as a practice that is strongly wrapped up in ethics. How a developer designs their code, how careful they are to protect the data their code may be used with, and how severe a security breach can be are all factors which affect a developer's attitude and understanding towards their work.

Ethics and Productivity

Ethics plays a significant role on a worker's productivity. As we have seen above, a developer's mood can impact the speed and quality of their work significantly, as a study at Hitachi Limited has shown, "compared to people who are unhappy, people who are happy have 37% higher work productivity" (Yano et al., 2015).

Therefore, on an individual level, we see ethics playing a significant role in the speed and quality of a developer's work, as their mood is closely linked with the stresses and pressures they may feel when tackling ethical issues.

In a company and project-wide level, we see that holding to careful moral codes leads to happier employees and a greater chance of success for the project or company. As a study published in Jim Collins and Jerry I. Porras' book "Built to Last" shows that companies that

tend to outperform in their industries tend to be the ones which exhibit “ethically correct” and socially responsible practices and norms (C. Collins and I. Porras, 1994).
Therefore, it is visible that on an individual and company level that good ethical practices are of high importance.

Efforts to Protect Data
In recent years, the ethics behind software and the internet has been the subject of much debate. In fact, 2018 marks the year that GDPR (or General Data Protection Regulation) came into act. The requirements to GDPR are “partly driven by high-profile data breaches” (Vaughan, 2018) and request that websites inform their visitors of the data that is being extracted from them and be willing to delete any information relating to a user upon request.
This will greatly improve the work of a software developer, giving them clear guidelines for how to treat data. As managing director of the Castlebridge information quality consultancy in Dublin, Daragh O Brien said in a recent webinar, “All the things GDPR asks you to do are simply good information management practices” (Vaughan, 2018).

Conclusion
When it comes to measuring a developer’s work, simplistic metrics are an option, but likely may cause harm to productivity and often don’t show useful results. A solution to this is to measure metrics like code quality and quantity in a variety of ways. This can be done with tools like the PSP and TSP or the Seven Tools of quality measurement. While these tools are time consuming to use, they provide valuable insight into a team or individual’s work ethic. On the other hand, we can measure a developer’s use of social tools, or perhaps even their physical well-being to understand better their productivity and ability to contribute to their work.
The algorithms used vary between studies but generally they create some sort of correlation between quantity, the number of lines written or hours worked, against quality, the number of bugs in the code or its importance in a project. This produces results that can be represented in a variety of graphs for ease of use.
Finally, a major concern is the ethics behind measuring a developer’s work and behind software development as a job. These issues have a variety of angles to them, on the one hand, an employee has the right to privacy, on another, an employer has the right to manage their own business. With that, a developer should consider their work carefully when designing and implementing it and consider the repercussions of them.

References

S. Humphrey, W. (2000). The Personal Software Process (PSP). *Software Engineering Institute*.

M. Johnson, P. (2013). Searching under the streetlight for useful software analytics. *IEEE Software*.

Hakes, T. (2018). *How to Measure Developer Productivity*. [online] 7pace Blog. Available at: https://www.7pace.com/blog/how-to-measure-developer-productivity [Accessed 25 Oct. 2018].

Cs.stanford.edu. (2018). *Crunch Mode: programming to the extreme - The Relationship Between Hours Worked and Productivity*. [online] Available at: https://cs.stanford.edu/people/eroberts/cs201/projects/crunchmode/econ-hours-productivity.html [Accessed 25 Oct. 2018].

Kanat-Alexander, M. (2018). *Measuring Developer Productivity » Code Simplicity*. [online] Code Simplicity. Available at: https://www.codesimplicity.com/post/measuring-developer-productivity/ [Accessed 25 Oct. 2018].

Hodges, N. (2012). *Measuring Developer Productivity*. [online] Nickhodges.com. Available at: http://www.nickhodges.com/post/Measuring-Developer-Productivity.aspx [Accessed 25 Oct. 2018].

S. Humphrey, W. (2000). *The Team Software Process*. [online] esources.sei.cmu.edu. Available at: https://resources.sei.cmu.edu/asset_files/TechnicalReport/2000_005_001_13754.pdf [Accessed 25 Oct. 2018].

Bughin, J. and Chui, M. (2013). http://www.nextlearning.nl/wp-content/uploads/sites/11/2015/02/McKinsey-on-Impact-social-technologies.pdf.

Yano, K., Akitomi, T., Ara, K., Watanabe, J., Tsuji, S., Sato, N., Hayakawa, M. and Moriwaki, N. (2015). Measuring Happiness Using Wearable Technology —Technology for Boosting Productivity in Knowledge Work and Service Businesses—. *Hitachi Review*, 64(8).

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

Stephen H. Kan. 2002. Metrics and Models in Software Quality Engineering (2nd ed.). Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Dario, J. (2018). *Measuring Developer Productivity – Hacker Noon*. [online] Hacker Noon. Available at: https://hackernoon.com/measure-a-developers-impact-e2e18593ac79 [Accessed 25 Oct. 2018].

Heile, J., Wright, I. and Ziegler, A. (n.d.). Measuring software development productivity: a machine learning approach. *Semmle Inc.*.

J. Thomson, A. and L. Schmoldt, D. (2001). Ethics in computer software design and development. *Computers and Electronics in Agriculture*.

Citizensinformation.ie. (n.d.). *Surveillance in the workplace*. [online] Available at: http://www.citizensinformation.ie/en/employment/employment_rights_and_condition s/data_protection_at_work/surveillance_of_electronic_communications_in_the_workplace.html [Accessed 25 Oct. 2018].

C. Collins, J. and I. Porras, J. (1994). *Built to Last: Successful Habits of Visionary Companies*. 1st ed. William Collins.

Vaughan, J. (2018). *GDPR requirements put focus on data ethics, governance*. [online] SearchDataManagement. Available at: https://searchdatamanagement.techtarget.com/feature/GDPR-requirements-put-focus-on-data-ethics-governance [Accessed 25 Oct. 2018].