Measuring Engineering – A Report

Rory Murphy

Introduction

Measuring the productivity of workers has been a field of study since the early 20th century. Pieces such as Fredrick Taylor's "*The Principles of Scientific Management"* and Frank and Lillian Gilbreth's motion studies established the methods by which manual worker productivity has been measured for the past 100 years. However, these processes of improving productivity by determining the optimal method of completing tasks and monitoring employees to ensure that these methods are strictly adhered to are no longer considered to be the definitive ways of measuring employee productivity. While the ethics of such meticulous probing of employees is discussed later, the primary reason for the failure of scientific management today is that it cannot be easily applied to knowledge workers. As pointed out by Peter Drucker, knowledge workers, unlike manual workers, are not expected to complete the same task repeatedly. Instead, they are required to constantly ask themselves "What is the task?" and determine the best way to complete it (Drucker, 1999). With requirements that vary from job to job and constantly changing requirements in said jobs, the scientific management methodologies have become irrelevant for knowledge workers such as software engineers.

Due to this complication, certain researchers argue that the measurement of software engineering productivity is a somewhat pointless endeavour. Articles such as *"The Myth of Developer Productivity"* argue that commonly used measurement methods such as lines of code and defect rates can simply be gamed by developers and restrict the innovation-oriented mindset required to thrive in the software engineering environment (Barnes, 2015). Others point out that "Much industrial metrics activity is poorly executed" (Fenton and Neil, 1999) and if data is poorly collected and analysed, how useful an analysis does it produce? Despite this, however, recent advances in measuring and analysing methodologies as well as developments in cloud computing technologies and automated data collection methods have led to the creation of more sophisticated data analytics techniques that, while not yet being perfected, can further advance the field of measuring software engineering, and other forms of knowledge worker productivity. With the variety of tools existing today, managers can collect more accurate data and utilise significantly more sophisticated analytical approaches to explore productivity.

The first segment report discusses the various types of data that can be measured, exploring the range of methods from simple manual data collection to the more complex data collected through automated processes. Next, the tools and techniques used to measure this data are explored, which is again divided into manual and automated processes, showing how the automatic tools and techniques can collect far more detailed and specific data than the previously used manual methods. After this the range of algorithmic approaches used to convert this data into meaningful and useful analyses of the processes and practices used by software engineers is addressed. Here, the value of automated data collection becomes apparent, with more complex algorithms using the

large volumes of automated data collected to develop more specific and relevant inferences about engineering productivity than could ever be made through manual data processing. Finally, while the development of data collection and processing methods has increased the amount of information available to managers about employees, there is a pressing ethical concern with regards to the collection and usage of this data that should not be ignored. Thus, this report also dedicates time to the discussion of this issue with the intention of ensuring that the ramifications surrounding this level of analysis are well known to the reader.

Section 1: Measurable Data

Measuring data is the groundwork for performing any sort of analysis and forms the basis of productivity measurement. Without collecting relevant data, effective analysis simply cannot be performed. This section discusses some of the more general measurements made of software engineers to track their productivity as well as some newer methods adapted by agile environments. The successes and failures of these is also explored as is the effect that the Cloud and other technologies has had on data collection. Finally, the data collected to perform some of the more advanced performance analysis methods is discussed.

One of the most basic measures of a software developer's productivity is the number of lines of code written. This method dates to the late 1960s (Fenton and Neil, 1999) and during the early years of software engineering was a commonly used measure. The flaws with using such a measure are clear to even a novice programmer, as it is easy to pad a program with additional lines to make it longer. Worse still, pursuing longer code leads to a variety of negative results such as less optimised programs and employees being afraid to remove lines even though doing so could make a program run more effectively or make it easier to understand (Barnes, 2015). Ultimately, large amounts of code can have a negative effect on a piece of software, and this is a prime example of a measurement that, while appearing to be relevant and useful, is more likely to hurt productivity when measured.

Other simple metrics such as the number of bugs closed, the number of defects per 1000 lines of code and time estimations made by developers are similarly problematic measures because, like lines of code, they simply promote less effective work habits. Barnes discusses these, pointing out that measuring the number of bugs closed promotes creating more bugs in the first place, measuring defects will make programmers avoid attacking more complex problems for fear of how it will affect their productivity measures and holding software engineers accountable for time estimations creates situations where the dominant strategy is simply to estimate longer as well as creating difficulty when a problem's definition and requirements inevitably change during the engineering process (Barnes, 2015). The common element connecting these measures is that they are simple to find and understand and appear to be relevant at a surface level. However, it does not take much exploration to show that using these methods to measure productivity can have a negative effect, causing developers to sacrifice effective coding practices and avoid difficult work to improve these measures. Johnson refers to this type of analysis as "searching under the streetlight",

whereby easy to collect data is analysed however its result are mostly unhelpful (Johnson, 2013).

There has been a recent movement in software engineering towards Agile development; a method that promotes faster response to changes and greater client communication to uncover "better ways of developing software by doing it and helping others do it." (Manifesto for Agile Software Development, 2001). As a result, it has been noted that different productivity measurement methods are required to adjust for these changes (Javdani et al., 2012). Javdani et al.'s paper refers to a number of techniques centred around breaking down a project into very specific tasks required for the project's completion and assigning them story points; measures of the amount of effort required to complete a specific task.

The following are the main metrics created using story points. Effort estimation; a team's estimation of how long each piece of work will take using story points to record the size of each relevant task with their sum being used as the measure of effort. Velocity measurement; the speed at which stories are being completed by individuals and the team which can be used to determine how productive members are relative to one another as well as be combined with effort estimation to determine an approximate time until completion of a project. Burndown charts which create an estimate of the amount of work to be done in a period and can then be compared to the amount of work completed for said period. Cumulative flow charts which show the amount of backlogged work yet to be completed. Response to change measures which measure the hours that have been spent reworking the project, which is used to indicate the project's ability to adapt to change.

These measures can be used to give a better picture of the size and speed of a project as well as the efficiency of each member in the project (Javdani et al., 2012). These measures are more focused than the previously mentioned methods, with clearer links between what they measure and how it affects a project. As well as this they are lightweight and well suited to the agile environment in ways that collecting large amounts of data manually are not suited to (Javdani et al., 2012). However, their dependence on story point estimations, which are based on employee estimates, severely reduce their effectiveness, and as pointed out by Barnes, result in inconsistencies from team to team and allowing developers to simply up the story point measure of their if their productivity is considered too low (Barnes, 2015).

The methods mentioned up to this point have relied on basic measures and individual judgement to analyse productivity. However, the advent of cloud computing, which allow for low cost data storage that can easily be accessed (Pocatilu, Alecu and Vetrici, 2010), as well as the rise in accessibility of automatic data collection tools (which are discussed further in the next section) has led to an increase in the availability of far more specific information about employees and their work habits. With this vast increase in measurable data there are now a variety of new methods that can be used to assess worker efficiency.

Hassan and Xie propose that software repositories should be mined for the rich data that they contain about software systems and projects. Software repositories are generally used as a record keeping system, however with the volume of data contained by them, they have the potential to greatly enhance software development decisions, providing a wellspring of relevant information about the work being done by engineers (E. Hassan and Xie, 2010). Now that companies can easily store this level of data about their employees on cloud services, there is easy access to a huge amount of analysable data.

Another approach to measuring software engineers is to measure their workflow (Snipes et al., 2013). By using automated data collection systems (which are discussed in the next section) to collect "fine-grained data" about the specific steps taken by software developers when performing tasks, managers can analyse these different steps and determine which methods are efficient and which are not. The information collected here can not only be used to assess which engineers are using more productive methods than others, but also to "provide information on how to improve and motivate the developer to adopt the new patterns in a positive way" (Snipes et al., 2013). Like repository mining, this approach would not have been feasible without recent technological innovations.

Dittrich, Gunes and Dascalu describe a method for identifying subject matter experts by using data obtained from the log files of repositories to create bipartite graphs connecting each author to the files that they have changed (Dittrich, Gunes and Dascalu, 2013). They propose using this to identify which engineers are knowledgeable about the different topics in a project; a useful metric when creating a better understanding of the productivity of the engineers involved in said project.

Wearable technology is another recent development in data measurement. One use of this technology is to measure employee happiness, a measure that "significantly influences performance", with a study showing happy workers to have 37% higher productivity and 300% higher creativity (Yano et al., 2015). By attaching wearable sensors to engineers, data about their physical activity and diversity in body movements, which are shown to be correlated with happiness, and thus productivity (Yano et al., 2015) can be collected. Wearable technology can also be used to determine how different members influence each other in a social system, leading to a better understanding of how well different engineers work with each other (Wei Pan et al., 2012).

It should be clear by now that recent technological advances have created an environment where the amount of data systems can collect has grown immensely. With automated data collecting systems to produce more data and cloud accessibility to store it, it is no surprise that the measurements taken of engineers has grown considerably in recent years. Now, employers can gain a more complete understanding of their employee's working environment to a very detailed and specific degree.

Section 2: Computational Platforms Available to Perform Work

Historically, data collection has mostly been manual, and a variety of techniques exist for performing and assessing these measurements. However, the development of automatic data collection has led to the creation of a variety of computational platforms that can collect and assess data more quickly and with less judgement-based analysis. In this section, the various manual and automatic data collection techniques are described as well as their benefits and drawbacks when compared to each other.

Generally, manual data collection requires using some process or standard to collect data about engineers and the projects they are working on, using the results to make inferences about the progress being made and productivity in these projects. Companies like IBM and Hewlett-Packard collect a number of key measures, with IBM using CUPRIMDSO (capability, functionality, usability, performance, reliability, installability, maintainability, documentation/information, service and overall) and Hewlett-Packard using FURPS (functionality, usability, reliability, performance, and serviceability) as different measures of a product's success, sending surveys out to customers rather than employees (Kan, 2014). These methods can provide an overall review of a product's success or failure but say little about the individual members of the team behind the product.

Another approach mentioned by Kan is the use of the Software Productivity Research (SPR) assessment and the Malcolm Baldrige National Quality Award Assessment. SPR assessment assesses 300 questions, providing a rating from 1-5 (with 1 being excellent and 5 being poor) with a focus on technologies and processes being used, the ergonomics and work environment of staff, and personnel and training in a team. As of 2000, SPR had assessed nearly 350 companies and 50 government organisations (Kan, 2014). The Malcolm Balridge National Quality Award Assessment is a quality award given in the US for quality in approach, deployment, and results. A report is given to companies highlighting areas of potential improvement (Kan, 2014). Both ideas mentioned here have the benefit of being performed externally, meaning company resources are not spent on analysing and producing findings, however the major drawbacks are that external reports will take a long time to perform and cannot be adjusted to suit the specific needs of management. Despite this, an outside opinion could present measurements and findings that had not yet been considered by management up to this point.

The personal software process (PSP) is another common method for manual data collection. It requires developers themselves to fill out 12 forms including a project plan summary, a defect-recording log and size and time estimation templates (Johnson, 2013). While this allows for a level of flexibility in what is being measured, the manual nature of PSP leads to incorrect conclusions being reached due to subjectivity of manual inputs (Johnson, 2013). As well as this, as with the other manual data processing methods explored here, it is time consuming (Sillitti et al., 2003). An alternative to PSP is LEAP, a tool that, while requiring some manual data input, automates some parts of the PSP process to reduce the errors associated with manual data inputs (Johnson, 2013). While LEAP improves the accuracy of PSP, it reduces its flexibility somewhat due to automation. Other prevalent process assessment approaches include Capability Maturity Model Integration (CMMI); a tool used for measuring process improvement on a scale from 1 to 5, ISO/IEC 15504 (SPICE); a set of

technical standards documents used in the software development process and ISO 9001; a set of standards relating to the quality of management systems (Grambow, Oberhauser and Reichert, 2013).

As shown above, manual data processing generally involves relying on the judgement of employees, managers, and external sources such as customers to assess the quality and efficiency of the systems being used. However, the common issues arising with these methods are their reliance on personal judgement, which can be inconsistent and varied, and the fact that manual data collection consumes valuable engineers' and managers' time. On top of that, as companies move towards using agile methodologies, less time is dedicated to the documentation and recording of work being done, which could further hurt the use of these practices. As a result, automatic data collection and computational platforms have begun to emerge. As mentioned previously, automatic data collection has led to the collecting of many types of data that could not previously be analysed, resulting in the use of productivity measures that simply would not have been possible only a few years ago. Furthermore, with the volume of data being made available increasing, automated tools are a requirement for dealing with this level of information. While there are drawbacks to these tools, the evidence suggests that a much greater movement towards their use will occur in light of the previously mentioned benefits.

As mentioned previously, the automation of LEAP resulted in some loss of flexibility in the PSP process (Johnson, 2013). However, this problem is mitigated by modern tools that collect a greater volume of data like Hackystat. As a tool, Hackystat was built to include four major features; the ability to collect both client and server-side data, the ability to collect data unobtrusively, a focus on fine-grained data collection and a focus on both personal and group-based development (Johnson, 2013). Hackystat's Software ICU then displays a variety of measures such as DevTime, commits made and code complexity measures based on the data collected. Hackystat is a popular measurement tool, being used in several measurement methods such as Zorro, a system for recognising test-driven development practices (Kou, Johnson and Erdogmus, 2009), and work pattern analysis (Snipes et al., 2013).

Another commonly used tool for collecting and analysing software metrics is PROM (PRO Metrics). A similar technology to Hackystat (Johnson, 2013), PROM collects information at personal, workgroup and enterprise levels to preserve developer privacy while assisting managers in analysing productivity as well as perform activity-based costing (Sillitti et al., 2003). PROM differentiates itself from Hackystat by monitoring and analysing the entire development process as well as the improvement of performance for individual developers while Hackystat focuses on the latter alone (Sillitti et al., 2003). The work pattern analysis method described by Snipes et al. also recommends using PROM as a method for collecting data.

A tool proposed by Javdani et al. for analysing the agile method data described in section 1 is COSMIC. COSMIC published an official version with specific agile modifications in 2011 and is used to estimate the size of a software project using story points, differentiating itself

by adapting to changes requested by customers, adjusting the size of each user story to include changes to previously released stories (Javdani et al., 2012). For a company recording the agile measures described by Javdeni, the COSMIC tool can be used to assist in software size measurement should this be required, however this does little to improve the problems described earlier with relying on such measures, as measurements used still heavily rely on individual estimation.

Zorro, as mentioned previously, is a tool that is used to recognise when developers are complying with "an operational definition of Test-Driven Development practices". By gathering data collected using Hackystat (such as whenever a developer invokes a unit test or edits production code) and applying SDSA (Software Development Stream Analysis), a Hackystat application used to define the rules and analyses necessary to interpret the TDD methods used by engineers (Kou, Johnson and Erdogmus, 2009). Zorro is an example of a tool with a more specific purpose than those previously mentioned but highlights the power of automated data collection tools. The approach Zorro uses is discussed in more detail in section 3.

The methods of measuring physical data via wearable technology requires its own set of computational platforms. Tools like the sociometric badge can record speech signals such as enthusiasm, body movement and detect the relative position of other badges (and thus other wearers) sending out this information in real time via radio waves (Kim et al., 2008). This data can then be processed and analysed by Hitachi's "H" artificial intelligence, which can automatically generate key performance indicators based on the data received and determine what the major factors influencing an employee's happiness are (Yano et al., 2015). The H tool, like Zorro, is another example of the powerful results observed by automated data processing.

As with measurable data, the introduction of automation significantly increased the capabilities of the computational platforms being used to assess engineering performance. However, there are drawbacks associated with these tools. While the flexibility issue mentioned by Johnson is mitigated by simply collecting a greater volume of data, Johnson also compares the level of data measured by Hackystat to "the harsh glare of operation-room lights" (Johnson, 2013). He goes on to explain that measuring this level of data can create a negative atmosphere amongst developers, who feel uncomfortable with being monitored to this degree. The ethical implications of these measuring methods will be discussed further in section 4 however this downside is a significant factor in the effectiveness of automated data collection methods.

Ultimately, while automated computational processes provide more in-depth analysis of engineering productivity, allowing for more specific calculations to be made using far more data in far less time with little employee interruption, the social response to such methods is far from insignificant. On top of that, while judgement-based analysis can be unreliable at times, there is still a place for manual data processing, especially in relation to the feelings of management and clients, which remain the key factors on whether a project is deemed successful or not.

Section 3: Algorithmic Approaches Available

While manual data processing tools require very little algorithmic analysis, as with significantly less data of which much is judgement based, analysis is usually restricted to evaluating the mean values and confidence intervals of results seen from an entire team, with the implications of such studies being left open to the judgement of management. In contrast, one of the key strengths of automatic data collection and processing is that far more data can be analysed at a faster speed, allowing for more complex analysis to be performed which yield significantly more information. In this section, some data processing methods, which can be implemented using some of the previously mentioned computational platforms are explored and the value that they provide in relation to measuring engineering performance and practices is described. The aim is to highlight the benefits of collecting such large amounts of data in relation to engineering productivity.

Methods for better understanding the relationship between different engineers in a project and the knowledge and capabilities that they possess is undoubtedly a valuable metric for improving a team's productivity and several methods for using more recent computational platforms to explore these relationships have been proposed and are explored here. In section 1, a method for determining which engineers are subject matter experts in different topics by generating bipartite graphs from log data is briefly discussed. Dittrich, Gunes and Dascalu emphasise the importance of understanding such a metric as should a developer leave a team, management need to know as soon as possible who can continue their work or whether another expert in this field needs to be hired. As well as this, this analysis allows new members to identify who the best individual in a team to ask should they encounter a specific development problem. Moreover, the bipartite graph can also be used to identify core developers (those with the most connections in a project) and how connected two authors are to each other (through eigenvector centrality) (Dittrich, Gunes and Dascalu, 2013). These metrics can help to explore which engineers frequently work with one another.

Another approach to better understanding team dynamics is the use of influence models. Through using stochiometric badges and other physical data collection tools, different time series can be collected and assessed to determine the different degrees by which employees are influenced by one another (Wei Pan et al., 2012). By assessing the body language of each developer when interacting with different combinations of other team members and comparing them, a prediction can be made about how influential individuals are on each other's actions. Not only that, but this model incorporates machine learning, allowing for this network of influence to be dynamically updated as interactions between developers alter their perceptions of one another. As with other machine learning algorithms, sufficient training data will need to be collected before the model becomes accurate, however once a strong model is developed, it will provide management with a far greater understanding of the different relationships between members in a project, allowing for them to take action where needed, with the results of said actions being displayed by the influence model through its dynamically adapting algorithm. Peer interaction is one of the most effective methods for developers to learn new tools and improve efficiency and

that currently it is shown to occur too infrequent (Murphy-Hill and C. Murphy, 2011), so ensuring that members of a team have a good relationship can improve the likelihood of this occurring and have a positive effect on efficiency.

Using wearable technology to improve productivity does not stop at team member interactions. As discussed previously, employee happiness can greatly increase productivity and creativity, and with wearable technology and Hitachi's "H" tool, by measuring physical activities correlated with happiness, key performance indicators associated with these activities can be generated and produce evaluation functions that predict individual employees' current happiness (Yano et al., 2015). With this information, the work environment can be optimised to increase happiness. For example, these measures could be used to determine the optimal room temperature for the employees currently in a given room and the air conditioning system could be connected to this tool, automatically changing to adhere to this requirement (Yano et al., 2015).

Tools like Hackystat and PROM, which collect large volumes of data about the specific activities of software engineers, can be used to provide relevant data to a variety of different algorithms used to assess and improve productivity. As mentioned in section 2, work pattern analysis can be performed using the data collected from tools like Hackystat and PROM. With the collected data, a Markov chain of steps taken during the development process can be built and used to identify similarities between different developers' processes and any unnecessary steps being taken emerge (Snipes et al., 2013). From this, developers can learn from the effective and ineffective methods being used to improve productivity.

These tools can also be used in conjunction with other tools such as the previously mentioned Zorro tool. By using Hackystat and SDSA to collect streams of low level developer behaviours such as invoking a unit test or editing production code and partitioning these streams into development "episodes", a rule-based system can be applied to assess how compliant these episodes are with test-driven development practices (Kou, Johnson and Erdogmus, 2009). The outputs produced by Zorro show why a specific episode was or was not classified as TDD conforming allowing for developers to adapt and improve their own practices. While Zorro is built for use with the Java programming language on the Eclipse IDE, the principals of this tool could be applied to any test-driven system.

The approaches described here are just an example of the multitude of algorithms that can be used to analyse the productivity of software engineers. Even then, the information gained from these methods alone can track several important productivity metrics and provide clear and relevant ways to improve these processes. On top of this, the development of algorithms based on automatic data processing is a relatively new field, and the current approaches are likely not the final iteration of productivity measurement using automatic data collection and processing. For example, Hassan and Xie propose that data mining algorithms and outlooks are a potential source of techniques for analysing software productivity data, even stating that working with the data mining community to develop algorithms specifically intended for assessing productivity could be an option (E. Hassan and

Xie, 2010). However, the backlash to this level of employee monitoring was briefly mentioned in section 2, and such a thing cannot go unignored as data privacy has become a very prevalent topic in modern society.

Section 4: Ethical Concerns

Data privacy has quickly come to the forefront of discussions about modern technology, with controversies like the Facebook–Cambridge Analytica data scandal leading to widespread conversation and concern about the topic. As mining and processing data becomes increasingly more effective, the fear individuals have of their data being used against them or leaked grows. In this section, the effects of measuring large volumes of data on employees is discussed, with the positives and negatives being outlined. As well as this, the recent implementation of General Data Protection Regulation (GDPR) in the EU is used as a framework for discussing how the current backlash towards the collecting of private data is being addressed.

Throughout this article, a variety of productivity measures have been discussed. In a perfect world, management can use these tools to benefit themselves and their employees. Analysing work patterns can be used to reward the use of efficient patterns, and show developers using less efficient patterns where they can improve (Snipes et al., 2013), measuring employee happiness can be used to discover how a working environment can be improved to increase engagement and work satisfaction (Yano et al., 2015) and identifying subject matter experts in a team can help new members to identify who to ask should a specific problem arise (Dittrich, Gunes and Dascalu, 2013) allowing them to more quickly and easily integrate themselves into the team. On top of this, these techniques create more effective and accurate results, meaning that as they replace older methods, employees that perform effectively will be better rewarded while those facing difficulty will have an easier time determining how they can improve, rewarding hard work and learning.

However, the prevailing outlook towards measuring large volumes of data appears to be negative. As mentioned previously, Johnson compares the level of detail of the data collected by Hackystat to “the harsh glare of operation-room lights”. While the data collected by tools like Hackystat allows for relevant and effective inferences about productivity, the fact that these tools collect data without telling users what it is they are collecting creates a sense of unease among employees (Johnson, 2013). While tools like PROM are more focused on the overall development process, sending only aggregated data about employees to managers (Sillitti et al., 2003), other tools do not necessarily follow this example. With managers having access to such detailed information about the performance of their employees, problems could arise in the working environment, namely using this data to over work employees, publicly shaming them with performance metrics should they not do so or to create a more difficult working environment for an employee that they dislike.

Amazon is a company renowned for its efficiency as well as constantly monitoring its employees. An article published by The New York Times explored the working environment

created by this. Amazon promotes an environment where “The workplace should be infused with transparency and precision about who is really achieving and who is not.” (Kantor and Streitfeld, 2015). A former employee was quoted saying “workers are encouraged to tear apart one another’s ideas in meetings” while management data is used to “spur its tens of thousands of white-collar employees to do more and more.”, resulting in another former employee saying they saw fellow workers who “practically combust.” (Kantor and Streitfeld, 2015). This situation encompasses the fear employees have about their performance data being used to create unhealthy working environments, allowing managers to use performance data to create hostile competition between employees, who work ceaselessly to keep up with each other and not be seen to be performing the worst. Such a situation could easily arise using some of the performance metrics discussed in this paper should managers choose to deliver the performance metrics in such a fashion.

Another issue associated with modern productivity measurement methods is the privacy of certain kinds of data. This issue becomes especially prevalent in the measurement of physical data such as happiness and employee relationships. The measurement and monitoring of such metrics are more personal than monitoring working practices, as these metrics, while related to work productivity, are more connected to the personalities of individuals than how good they are at complying with test driven development practices or utilising efficient work patterns. In the best-case scenario, managers can observe these metrics and openly discuss them with individual employees, ideally creating a more satisfying working environment for them or mending a difficult relationship that exists between two employees that managers were made aware of through tools like Hitachi’s “H”. However, this type of data again gives managers the potential to make certain employees’ lives more difficult should they choose to, creating uncomfortable working environments and teaming them with workers that they may not perform well with. Once again, the level of power held by managers in this case may be too high.

Management authority is not the only issue held by developers that are subject to such productivity measures and the security of their data is an important consideration. When using the cloud to store data, careful consideration must be made as to where the data is stored, how it is encrypted, who has specialised access to it and what happens in the case of a data breech (Pocatilu, Alecu and Vetrici, 2010). While cloud systems create more security than internal storage, making it difficult for thieves with a specific interest to locate the data and with the data being monitored by companies with a specific goal of protecting it (Pocatilu, Alecu and Vetrici, 2010), should a data breech occur the ramifications can be great. The Facebook-Cambridge Analytica scandal, which saw the data of “up to 87 million people” being harvested (BBC News, 2018) resulted in a massive drop in confidence in the Facebook’s brand, and while this was a leak of public user data, a leak of employee data could lead to a similar loss in confidence of employees with the company and greater difficulty in attracting new workers to join.

GDPR shows how important data protection and privacy has become and the backlash that has emerged due to various data leaks and scandals that have occurred in the last decade. With the right to privacy being recognised as “a fundamental right that is essential to the

well-being of every human being" (Legal ICT, 2018), managers need to be wary about the data they measure. On top of this, GDPR requires "the use, purpose, and method of processing [personal data] must be made clear to employees" and that the details of such measurements, the security behind them and the specific purpose of these measures must be made clear to employees. GDPR is designed to, among other things, protect the privacy and interests of employees against the unfair use and storage of their personal data.

Ultimately, the measuring of fine grained data leads to a difficult trade off; collecting more data allows for a better understanding of the working environment, however this greater understanding can be manipulated for better or for worse and can create a feeling of discomfort for employees. The main concerns associated with detailed data collection and measurement are the power it can grant to managers and the security concerns associated with the storage of such data. However, in my opinion the backlash towards such data collection is well founded. At the same time, these techniques pose significant and relevant ways to measure and improve engineering productivity, replacing less accurate metrics that have been used in the past, and while these upsides could improve the productivity of software engineering around the world, failing to implement them correctly will create a more hostile environment as shown by Amazon where managers are able to pit employees against each other and force them to work much harder and for much longer than should be expected of them. The regulations imposed by GDPR are clearly designed not to prevent the collection and processing of employee data but instead to ensure that the data collected remains in the control of the employees who own it, preventing its abuse by more senior employees and ensuring the necessary security precautions are maintained. Such an outlook to me is necessary to ensure that these methods are correctly implemented, reducing the fears held by workers that their data will be abused by controlling the people analysing and acting on the inferences made by the data. This does not exclusively benefit employees, however, as if people become more confident that the data they give will be handled correctly and used to help them, they will be more willing to give it away, reducing the backlash Johnson describes that exists towards the collection of this type of data.

Conclusion

This paper has discussed the development that has occurred in the data used to analyse software engineer productivity as well as the chief reasons for this development. Several different platforms that are used for the processing of this data were also explored, as well as some of the algorithmic approaches used to analyse the data collected by these platforms and the ways in which this data can be used to measure and increase productivity. Finally, the major ethical concerns associated with the measurement of such fine grained data was explored, as well as what causes these concerns and how they are currently being addressed with the goal of creating a situation where performance metrics can be used to benefit the world of software engineering as a whole, from the perspective of both individual developers and industry productivity.

With the methods discussed here, it should be clear to readers that measuring software engineer productivity is a tangible objective, with modern technology being able to collect and process enough data not only to make relevant inferences about the effectiveness of

individual developers and teams but also to recommend ways to improve productivity. As well as this, while there are well founded concerns about the ethical implications of measuring this level of data, by introducing safeguards like GDPR to protect the rights of data owners and ensure the valuable knowledge produced by these techniques is not abused, measuring productivity can be safely used to improve the world of software engineering.

Bibliography

- Barnes, D. (2015). *The Myth of Developer Productivity*. [online] Dev9. Available at: https://dev9.com/blog-posts/2015/1/the-myth-of-developer-productivity [Accessed 17 Oct. 2018].

-BBC News. (2018). *Facebook fined £500,000 for data scandal*. [online] Available at: https://www.bbc.com/news/technology-45976300?intlink_from_url=https://www.bbc.com/news/topics/c81zyn0888lt/facebook-cambridge-analytica-data-scandal&link_location=live-reporting-story [Accessed 6 Nov. 2018].

-Beck, K., Grenning, J., Martin, R.C., Beedle, M., Highsmith, J., Mellor, S., Van Bennekum, A., Hunt, A., Schwaber, K., Cockburn, A., Jeffries, R., Sutherland, J., Cunningham, W., Kern, J., Thomas, D., Fowler, M. and Marick, B. (2011). Manifesto for Agile Software Development. Available at: http://agilemanifesto.org/ [Accessed 31 Oct. 2018].

- Dittrich, A., Gunes, M. and Dascalu, S. (2013). Network Analysis of Software Repositories: Identifying Subject Matter Experts. [online] Available at: https://pdfs.semanticscholar.org/d6b0/b5abe47a782ad0866f524e1e5b3e10d49a89.pdf [Accessed 11 Oct. 2018].

- Drucker, P. (1999). Knowledge-worker productivity: the biggest challenge. *California Management Review*.

-Fenton, N. and Neil, M. (1999). Software metrics: successes, failures and new directions. *Journal of Systems and Software*, [online] 47(2-3), pp.149-157. Available at: http://www.pauldee.org/se-must-have/new-software-metrics.pdf [Accessed 11 Oct. 2018].

- Grambow, G., Oberhauser, R. and Reichert, M. (2013). Automated Software Engineering Process Assessment: Supporting Diverse Models using an Ontology. *Int'l Journal on Advances in Software*, [online] pp.213-224. Available at: https://pdfs.semanticscholar.org/9a9d/5251aa571aa31cb30d01e1bf2464364ba08c.pdf [Accessed 24 Oct. 2018].

- Hassan, A. and Xie, T. (2010). Software Intelligence: The Future of Mining Software Engineering Data. *FoSER 2010*. [online] Available at: https://people.engr.ncsu.edu/txie/publications/foser10-si.pdf [Accessed 25 Oct. 2018].

- Javdani, T., Zulzalil, H., Ghani, A., Sultan, A. and Parizi, R. (2012). On the Current Measurement Practices in Agile Software Development. *Journal of Computer Science Issues*, 9(4), pp.127-133.

- Johnson, P. (2013). Searching under the Streetlight for Useful Software Analytics. *IEEE Software*, 30(4), pp.57-63.

-Johnson, T. (2018). *The Real Problem With Tech Professionals: High Turnover*. [online] Forbes. Available at: https://www.forbes.com/sites/forbesbusinessdevelopmentcouncil/2018/06/29/the-real-problem-with-tech-professionals-high-turnover/ [Accessed 6 Nov. 2018].

- Kim, T., Chang, A., Holland, L. and Pentland, A. (2008). Meeting Mediator: Enhancing Group Collaboration using Sociometric Feedback. *Proc. ACM 2008 Conf. Computer Supported Cooperative Work. New York: ACM*, [online] pp.457–466. Available at: http://vismod.media.mit.edu/tech-reports/TR-621.pdf [Accessed 1 Nov. 2018].

- Kou, H., Johnson, P. and Erdogmus, H. (2009). Automated inference of test-driven development with Zorro. *Automated Software Engineering*, [online] 17(1). Available at: http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=956E1624304502A89128F90EAF854128?doi=10.1.1.69.4525&rep=rep1&type=pdf [Accessed 25 Oct. 2018].

-Murphy-Hill, E. and C. Murphy, G. (2011). Peer Interaction Effectively, yet Infrequently, Enables Programmers to Discover New Tools. *in Proceedings of the ACM 2011 Conference on Computer supported cooperative work*. [online] Available at: https://people.engr.ncsu.edu/ermurph3/papers/discovery.pdf [Accessed 23 Oct. 2018].

- Pocatilu, P., Alecu, F. and Vetrici, M. (2010). Measuring the Efficiency of Cloud Computing for E-learning Systems. *WSEAS TRANSACTIONS on COMPUTERS*, [online] 9(1), pp.42-51. Available at: https://pdfs.semanticscholar.org/0d6f/2e0ee9dac8e6d8682c05c12c1e2b7bc01b08.pdf [Accessed 24 Oct. 2018].

- Snipes, W., Augustine, V., Nair, A. and Murphy-Hill, E. (2013). Towards recognizing and rewarding efficient developer work patterns. *35th International Conference on Software Engineering (ICSE)*, pp.1277-1280.

- Sillitti, A., Janes, A., Succi, G. and Vernazza, T. (2003). Collecting, Integrating and Analyzing Software Metrics and Personal Software Process Data. *Proceedings of the 29th EUROMICRO Conference "New Waves in System Architecture*. [online] Available at: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.138.6806&rep=rep1&type=pdf [Accessed 25 Oct. 2018].

- Wei Pan, Wen Dong, Cebrian, M., Taemie Kim, Fowler, J. and Pentland, A. (2012). Modeling Dynamical Influence in Human Interaction: Using data to make better inferences about influence within social systems. *IEEE Signal Processing Magazine*, [online] 29(2), pp.77-86. Available at: http://fowler.ucsd.edu/modeling_dynamical_influence.pdf [Accessed 25 Oct. 2018].

- Yano, K., Akitomi, T., Ara, K., Watanabe, J., Tsuji, S., Sato, N., Hayakawa, M. and Moriwaki, N. (2015). Measuring Happiness Using Wearable Technology. *Hitachi Review*, [online] 64(8), pp.517-524. Available at: http://www.hitachi.com/rev/pdf/2015/r2015_08_116.pdf [Accessed 24 Oct. 2018].