Measuring Software Engineering

#### Euan Leith

The goal of measuring software engineering is to improve productivity. A firm might want to improve the productivity of their workers in order to improve their product or to reduce the time taken for production. An individual engineer might want to improve their productivity for similar reasons, as well as to increase their expertise in the area, and increase their employability. There may exist vague ideas as to how to improve productivity. But to do so with confidence and accuracy, one must employ a more scientific approach: it must be measured. With strict measurements of productivity; certain actions can be proven to be more or less productive; the productivity workers can be analysed and improved; and changes in productivity over time can be studied and understood. This essay will explore how this productivity is determined; what elements of software engineering can be measured; what tools are available to perform this measurement; what algorithms can be used to perform this measurement; and the ethics of this measurement.

### How can SE be measured?

So what is productivity? In order to measure it, it must first be defined. Otherwise no consistent research can be performed, or tools produced. The common definition of productivity is output per unit over input per unit (OECD, 2016). This can be understood more specifically as the quantity and quality of output per unit of quantity and quality of input. Quantity and quality together make up an object; only including one wouldn’t be representative of the object as a whole. There may be some ultimate output, e.g. profit for a firm, as well as sub-outputs, e.g. output of teams in a firm, which will relate back to the ultimate output.

The quantity of input and output can be measured relatively easily as the cost per unit time and the number of units produced respectively. But what are these units produced? The output might be some combination of different types of data, for example code written, pull requests, documentation written, communication with clients, etc. Each of these types of output will be a set of units, each with a degree of quality. Consider a company producing glass bottles and a software development team. The glass bottles company might use the number of bottles produced per unit time, while the software development team might use the number of lines of code. The former seems to be a much more accurate measurement of productivity than the latter. Why is this? Only one type of output is being taken into account in both cases. However, unlike bottle production, software engineering includes more types of output. But this isn’t the only reason: the quality of each unit also hasn’t been taken into account. How do we determine this quality?

It’s evident that an individual might have some intuitive notion of the quality of a product. This data can be useful to some degree, but brings with it a number of issues. Firstly, the individual might have certain personal biases, for example disliking the creator of the product, which corrupts their determination of quality. Secondly, one individual can process much less data relative to a computer. In many cases there is simply too much data for humans to manage, so if possible, using computers becomes a necessity. However, computers aren’t necessarily perfect for this task either. Until quality can be measured perfectly accurately by a computer, they may make incorrect quality judgements which would seem obvious to a human. Computers don’t have the ‘common sense’ humans have as they don’t ‘understand’ what they’re doing in the same way humans do. Thus until computers (or humans) can be perfected, both may be used in concert to help cover each other’s blind spots. But whether performed by a computer or a human, the nature of the data to be gathered, analysed, and presented must be known. Thus we must figure out exactly what defines the quality of a product.

But can the quality of these objects be strictly defined? An individual might be able to - consciously or subconsciously - discern the quality of a product, even in relation to other products. Assuming that there is no function of the human brain that is beyond the capacity of a computer or some other measurement device, then this process can be understood and quality can be defined and measured. But what constitutes quality might differ from person to person; how do we determine an objective measurement of productivity? We can’t, unless we are measuring it in relation to some goal(s) (OECD, 2016). A glass bottle is considered good quality in the common sense only where the goal is to produce uniform bottles that can be drunk from. This is an issue if we are to measure quality with any consistency; so some goal(s) must be chosen. Within the context of software engineering, the ultimate goal of a firm can be considered as profit. The goal of some sub-section might be to create a product that satisfies a customer, is maintainable, etc. The definition of quality will also differ depending on the product, each of which will be more or less difficult. For example, the quality of a glass bottle may be relatively easily found to be determined by whether or not its broken or misshapen (in line with being uniform and drinkable), whereas the quality of a line of code is a more nebulous object. Just because quality can be defined doesn’t mean figuring out this definition is easy, or even practically possible. In section 2 we will explore tools which attempt to measure this quality, as well as that of other software development data types.

Once productivity is defined, the goal is to find ways to increase it. To do this, we can both; measure the quantity and quality of input and output directly; and find and measure phenomena which correlate with increases in quantity and quality of output. Once measured, this information can be analysed over time and acted on appropriately.

We’ll look at the different possible types of output to be measured. There are a number of data sources from which outputs can be analysed. In the field of software development, we will look mainly at repositories. Although there are others such as documentation and client communication, these have been explored relatively less as we will see in section 2.

Firstly, initial attempts at measuring software development were often simply to measure the number of lines of code written. This was a wildly inaccurate measurement of productivity mainly because it ignored the quality of each line. Measuring only the quantity of data effectively considers each element as having the same quality. Therefore in cases such as this where quality differs greatly between elements, quality must be taken into account. Furthermore just measuring quantity could be easily gamed by simply stretching out the code written over more lines. This is equivalent to reducing the quality of each line, while increasing the total quantity. Therefore this problem can be alleviated with a more accurate measurement of quality. So how do we determine quality? For an individual with knowledge of programming what a line is doing, measuring quality might seem intuitive. But there are too many lines of code in most products for an individual to go through each and assign a quality. So the process must be automated. There are simple lines which can be programmed to be discarded such as empty or duplicated lines. But gauging the quality of a standard line of code directly is very, very complicated. This process requires an understanding of the programming language and the rest of the system, in order to parse what a line is doing and how complex this task is. Effectively, the computer would have to understand how to code, in which case human programmers would become obsolete. So other approaches were taken. Data such as the number of locations new code was written in, and how it was changed over time were measured. This data, instead of directly measuring code quality, finds metrics which correlate with higher quality. If new code was written in multiple locations, it’s more likely to be changes to old code than new code, which correlates with higher quality. If code is being repeatedly re-written, each iteration is likely of lower quality than code written only once. As more and more of these correlative factors are measured, we will get a more accurate measurement of quality.

Next, we can look at the commits to a repository. We can measure the number of commits made, but this is even less representative of productivity than the number of lines of code. Each commit isn’t necessarily indicative of anything beyond that the developer is saving their work. This may be after doing a lot, or almost nothing. Furthermore this will differ amongst developers due to personal preference, and give an unfair benefit to those who happen to commit frequently. This may also result in those other developers gaming the system by simply committing more frequently. To rectify this would require a relatively more accurate measure of quality. But each commit is made up of lines of code, so the quality of commits can’t be measured more accurately than the lines that make it up.

The next piece of data we can get from repositories is pull requests. The quantity of pull requests will positively correlate with how much work needs to be done, as more pull requests means more code to be reviewed. But again, some code may be more complex to review than others, so quality must be taken into account. Similar to lines of code, analysing the quality of the request itself is very hard. However, unlike lines of code, the number of requests is likely small enough so as to be manageable by humans. In fact, the developer’s job here is to determine if the quality of the code is sufficient. Therefore the process can in large part be left up to the engineers. Because of this, automatic measurement of pull request quality is sparse.

Finally, issues raised can be measured. More issues means more work, so the quantity of issues negatively correlates with the amount of productivity required. Similarly, the number of issues resolved will positively correlate with the amount of productivity already expended. But the amount of work required to resolve an issue will differ, so quality (i.e. how easy or hard will the issue be to resolve) should also be taken into account in order to improve accuracy. Similar to pull requests, analysing the quality of the issue itself is very hard, but there are a small enough number of them that this process can be performed by engineers. However as mentioned, this approach does have its issues, so other factors can also be measured automatically to improve accuracy and save time. A possible approach is to measure the quality of the issue by analysing the tone of the issue comments. If certain emotions expressed here correlate with a change in productivity, there are two possibilities. If the emotions cause changes in productivity, then we can attempt to improve the emotional wellbeing of our developers in order to improve productivity. If the difficulty or quality of an issue causes changes in emotions, then we can gauge the quality of a given issue by the sentiments expressed. This approach has a number of issues however. As with all studies, the data collected isn’t necessarily representative of the wider population. Furthermore if the data isn’t collected from a software development firm specifically, it may not be representative of this type of firm. These can be improved with increased sample sizes and studies specific to the field in question respectively. Also, the tools used to determine emotions might be generally inaccurate, for example not noticing sarcasm.

In order to find factors which correlate with increases in productivity, data on productivity and potentially related factors can be collected and analysed.

There are a number of issues to this approach. Firstly, the data collected and processed isn’t necessarily representative of the wider population. This can be improved with larger sample sizes. Secondly, research which only finds correlations between phenomena still requires some causal link and direction to be found. Without this, it’s not been proven whether altering some correlative factors will in fact have the desired influence. Thirdly, mistakes on the part of companies using this information must be noted. A company might act on inconclusive data; and even where a study is perfectly accurate, the company acting on that data might act disproportionate to the degree of correlation. For example, if a relatively small correlation is found between the temperature of the workplace and productivity, a manager might go overboard by querying workers about it several times a day. Fourthly, studies performed on non-software development firms might not be representative. There may be relevant differences which should be taken into account, and where possible, studies should be conducted on software firms specifically.

However there are also a number of benefits. Firstly, there are certain factors which might intuitively correlate with productivity, such as equipment quality. But if the degree of correlation isn’t known, the amount of money and time spent on improving these factors will be decided solely by the personal inclination of individuals. The issues with this approach have been explored previously. Therefore if analysis is performed to obtain a more accurate measurement of the degree of correlation, these factors could be altered more accurately in accordance with this information. Secondly, factors which might not intuitively correlate with productivity, and were thus previously unknown, can be discovered. The more correlative factors that are known, the more that can be acted on, and the more productivity can be increased.

With these issues and benefits in mind, we can look at the studies and other analysis performed in this area.

Firstly, there have been a number of studies performed on the relationship between social networks and productivity. They have shown correlations between productivity and co-worker productivity (Lindquist, Sauermann, Zenou, 2015), sense of individuality in a group (Worchel et al. 2011), coordination and communication across software development teams (Catalda and Herbsleb, 2013), and many other phenomena. For each of these factors to be acted on, data must be collected. Therefore there is incentive for employers to gather as much general networking data as possible from their employees.

Secondly, studies have found a correlation between happiness and productivity in the field of software engineering (Graziotin and Fagerholm, 2019). If happiness causes an increase in productivity, it would be in the firm’s interest to improve it. This can be achieved in many ways, such as organised events, and potentially even reduced working hours. Notably acting on this data is directly beneficial for the employees as well as the employer, where others aren’t necessarily. If productivity causes an increase in happiness, this only further incentivises companies to improve their employees’ happiness to create a positive feedback loop of increased happiness and increased productivity.

There are many other factors such as fitness level (Cox, Shepard and Corey, 1981), temperature changes (Niemela et al. 2002), and listening to music without lyrics (Shih, Huang and Chiang, 2012) have been shown to correlate with productivity. Many of these phenomena have relatively small correlations with productivity, so employers can perform softer actions for them such as recommending exercise, asking employees about the temperature, and allowing employees to listen to music without lyrics. Here we can see that not all productivity measurement requires complex tools and strong actions; quick, cheap changes can also have an impact.

### What platforms can be used to gather and process data?

There are a number of different tools which can be used to gather, process, and present data from different sources. Their goal is to measure either direct output data for quantity and quality, or indirect data which correlates with increased productivity. Firstly, we’ll look at tools which directly measure output.

Within the field of software engineering specifically, there is a notable focus on code output, despite the existence of other significant outputs such as documentation and client communication. Whether this is because these outputs are harder to measure, because they’ve been neglected as legitimate outputs, or some other reason, this gap should be noted.

So now let’s look at the tools which measure data from repositories. Contained here is the code of the engineers, as well as other data such as commits and pull requests performed. We will look at four popular tools in this area; PluralSight Flow, GitClear, PinPoint, and Code Climate. Each of these provide some different metrics and functionality, but there is significant overlap. The total number of metrics between these tools is over one hundred, so we’ll analyse the metrics each company has listed as most important (Harding, 2020).

The differences between these tools can be seen mainly along two axes; the amount of data processed by the tool or by the user; and the number of data types measured. PluralSight Flow and Code Climate take a more scattershot approach: query a number of different types of output, with relatively little processing. This leaves a large part of the decision making up to the user. By contrast GitClear mainly focuses on a single data point in relatively more detail: ‘Line Impact’, which measures the quality of lines of code. PinPoint lies in the middle of user/tool processing, while presenting the data mostly in terms of pull requests. As previously explained, the accuracy of productivity measurement increases with an increase in the accuracy of the measurement of quality, and with an increase in the number of data types measured. GitClear is better with regards to the former and worse with regards to the latter, and vice versa for PluralSight Flow and Code Climate, while PinPoint is somewhere in the middle. Whether or not one of these is better than another will depend on the degrees of these attributes, their relative weights, and the goals of the customer.

Many of these metrics measure the quantity of output, which, as previously explained, isn’t an entirely accurate measurement of productivity. They will also be more or less accurate for different types of data, in proportion to the variance of quality as previously explained. Thus metrics measured here such as pushes per day and cycle time should be considered proportional to their degree of accuracy. However the degrees of accuracy are at least not easy to find, if not entirely unknown, in which case the consideration will end up being nothing but the personal inclination of the user. Most of these quantitative metrics are fairly simple, for example the number of lines of code or the number of ‘coding days’ (days where at least one commit was made). However we will now look in more detail at some of the more interesting metrics.

Firstly, the metric ‘code churn’ exemplifies a number of interesting properties. This metric refers to code which is repeatedly rewritten over a short period of time. The main thing of note about code churn is that it can represent a number of different things. If it’s at a high level, it might be that the developer is having trouble with a problem. This might be because the developer is stuck or inexperienced in that area, in which case help could be provided. But it could also be because the requirements were unclear, in which case communication with the client should be undertaken. But it might instead be that the requirements are quickly changing, where again one should consult with the client. It might be that the developer is prototyping or exploring a new idea, but it might rather be that the developer is polishing the work. In both of these cases, no action is necessary, but it might be useful to know which is being done. From these examples we can see that while code churn is a useful metric to measure, what it actually represents isn’t immediately obvious. This results in a number of issues. Firstly, tools which measure code churn as a metric which negatively correlates with productivity might be inaccurate. Secondly, looking at code churn on its own won’t reveal much actionable information; and some users might act inappropriately on the data despite this due to the unclear nature of the metric. So code churn isn’t a useless quantity, but shouldn’t be used in isolation, nor without caution.

Secondly, ‘impact’ is an attempt to measure the quality of lines of code. This is achieved in two parts. Firstly, by removing superfluous lines such as empty lines and comments. This gives a more accurate measurement of the quantity of lines. Secondly, the quality of each line is determined by correlative factors as previously described. Each of these factors will have some degree of relevance. For example, GitClear includes the programming language used, as well as the number of different locations the code spans. If Python is used, the impact of each line will be greater than if Java is used. Similarly, if the commit is only in one location, the impact of each line will be less than if it spans multiple. But what are the degrees of impact here, and how are they determined? Sadly, this information isn’t publicly available as far as I’m aware. Therefore this process can’t be fully understood here.

Next, we’ll look at tools which measure data which correlates with increased productivity.

There are many tools such as Isaak and Polinode which analyse the social network connections of companies. These tools provide the user with the knowledge of their organisational network, which can be combined with the knowledge of network factors which may correlate with productivity to determine potential actions to take. For example, remember that the productivity of a worker increases with the productivity of their co-workers. With this, if a worker is less productive than usual, and the network analysis tool shows that the people in their circle of communication are also less productive, the employer could link that worker with a more productive worker. This information can also help to determine which workers communicate across groups. These workers will likely be more involved in the higher level process, and so would be good candidates for promotion.

Next, as previously described, there is a positive correlation between worker happiness and productivity. If an increase in happiness results in an increase in productivity, it would be in the firm’s interest to increase happiness. But in order to do this, happiness must be measured for two reasons. Firstly, to see which workers are significantly unhappy, and thus should be helped. Secondly, to analyse the change in happiness over time in order to figure out both if the current approach is working, and if it does in fact result in an increase in productivity. There are two ways to measure happiness; questionnaires and tools. Questionnaires ask simple questions like ‘one a scale from one to ten how happy are you?’, and leave it up to the user to give an accurate representation of their wellbeing. This method is decently effective, but can be time consuming, not exhaustive, and the user might not be accurate. Automatic tools could be used to overcome these issues. Though not currently as common as questionnaires, we can look at one possible approach to automatic measuring. Hitachi produces a tool which measures factors which have been found to correlate with happiness (e.g. amount of sleep, physical activity) with a specific device (Yano et al. 2015). This data is translated into happiness data, from which the employer or worker can interpret possible actions to take to improve productivity. For example, if a worker is especially unhappy, the employer could give them the rest of the day off so that they return the next day happier, and thus more productive. The reason this approach is less common is that there are conflicting definitions of happiness. Until this is resolved, there will be no objective object of happiness with which other factors can be found to correlate.

Finally, all measurements of productivity are with respect to time, so how do we measure the time spent working? A simplistic approach would be just to take the number of paid hours as the quantity of time. But this doesn’t provide much information for improvement. So there are a number of tools which measure the quality of time, i.e. for how long a user is doing a specified task (e.g. Clockify, Timeular). This information can be analysed to discern if disproportional time is being spent on certain activities. This proportion should be determined by the quantity and quality of the task: a bigger, harder task should take longer than a smaller, easier one. There is one main potential issue with this approach. The user might inaccurately judge the proportion of time which should be allocated for each task, especially if this judgement is based solely on personal inclination.

### What algorithms can we use?

This section will explore in more detail specific algorithms and techniques for measuring productivity. The goal of these algorithms is to find causations between certain actions and productivity both directly and indirectly. If there is a positive relation of significant degree, the action should be taken, and vice versa. Firstly, we want to find which factors correlate with productivity, then apply that information.

It’s hard to find out how the tools for measuring productivity such as Code Climate determine which metrics to include, and where one metric is an amalgamation of other metrics (e.g. impact), how the weights of each element are determined, as this information isn’t public. This may be determined by some processing involving finding factors which correlate with other factors which have already been determined to correlate with productivity. It may also be determined by personal inclination. This information is unknown, therefore for the purposes of this essay we won’t look at how this information is determined. Instead, we will focus on the application of that information, i.e. how productivity metrics might be measured.

The data must first be collected. If the data isn’t collected by the analyser, it must be accessed from an existing data source. If there is an existing API for this source, the data can be accessed from there. Otherwise, the source will have to be manually scraped. After the data is collected, it must be cleaned before it’s usable. This might include removing useless or duplicate data, such as empty or repeated lines of code, etc. Once the data has been cleaned and is thus usable, it can be processed to produce meaningful information.

As previously described, most of the data collected from mining repositories is quantitative information such as the number of commits. These metrics require little processing once the data has been cleaned. They may be processed over time, for example ‘code churn’, i.e. how many times a commit has been altered over a short period of time. They may be processed with respect to quality, for example ‘efficiency’, i.e. the number of lines of code over the quality of that code.

A more complex process could be performed on the data to determine outliers. Consider a worker who has an unusually low number of commits per day relative to their co-workers. If this only occurred one day, maybe it could be disregarded as a random fluctuation. However if this persists, it may be indicative of an issue that should be resolved. Therefore knowing the existence of certain outliers is valuable information. In order to find these outliers, we can use a density-based clustering process. Here, the dense values will end up in one cluster, while leaving out the less dense values, i.e. the outliers. We will look at one specific density based algorithm: ‘DBSCAN’. A desired radius for a data point and a minimum cluster size are specified by the user. For each data point, if another data point is within its radius, it is added to a set called its ‘neighbourhood’. If the size of its neighborhood is greater than the minimum cluster size given, this neighborhood becomes a cluster. For each element of this cluster, if its cluster contains data points not in the original cluster, the two clusters may be joined together. The result will be one or more clusters of dense data, and a number of loose data points, which represent outliers.

Clustering can also be used to measure the social networks of a firm. A density based approach can be used again to find any outliers: workers that might not be communicating with their co-workers. But we can use different approaches to further understand this data. We can use partitioning to find groups of workers who communicate frequently with each other. We will look at one of the most common of these algorithms: ‘k-means clustering’. The goal is to separate the data into k clusters. Initially, the centres of these clusters are chosen randomly. The distance from each centre to each data point is calculated (in this case the level of communication between employees), and each data point is assigned to the closest centre. For each cluster, the mean of its data points is calculated, and this becomes the new centre. This is repeated until the mean calculation doesn’t change the centre. The data will now be partitioned into k clusters by distance.

We can also use hierarchical clustering to determine which workers communicate across groups, which can help to show which workers should be promoted as previously described. We will look at a common hierarchical clustering algorithm: agglomerative clustering. The idea here is to create clusters of increasing size which represent smaller groups of communication and how they interact with each other. First, compute the distance (level of communication) between each data point. Then, create clusters from the closest data points. Repeat for clusters of increasing size, until there exists one clustering containing all of the data points. From this structure we can see which workers connected larger clusters together, i.e. which workers communicate most across groups.

### Is this ethical?

There remains one glaring question: should we even do this? For the purposes of this essay we will consider this question from the perspectives of the groups involved; the employers, the employees, and members of society generally. An individual might be part of multiple of these groups. The goals of each of these groups might differ, in which cases different actions should be taken. The goals of each individual in each of these groups may differ as well due to personal inclination, but we will consider the goals of each individual solely as they relate to these groups, both for the sake of simplicity, and because these personal inclinations are here unknown. Thus an employer only considers profit; an employee considers both their own career interests and career-related interests such as time spent working; and a member of society generally considers only typical interests such as health. In this context each member of a group can be considered as having the same goals. For each action taken to measure productivity, there will be a corresponding degree of positive/negative motivation for each group. There may be intuitive, vague notions of these degrees, but this essay won’t attempt to describe these, nor give strict values for them. Some of these negative factors can be improved or fully alleviated, however others are inherent and thus unchangeable. For example, a tool which tracks employees’ happiness can be made anonymous, thus allowing the user to determine only the average happiness level of the firm. However the employee might not want this data to be collected at all. This can’t be done without stopping the measurements, so as long as they are being taken, this negative factor will exist for the worker. Finally, for each action, for each group, if the sum of positive motivations is greater than that of the negative motivations, then yes, they should do this.

Consider the positives and negatives of a change in the level of productivity generally. For the member of society, it comes down to a fairly simple calculation: the degree to which the increase in productivity increases the profit of those firms, which improves the economy, which improves their life as a member of that economy against the degree of sadness incurred by the employees and employers in measuring this productivity. Similarly for the employer, this is relatively simple: more productivity means more profit. Reduced productivity may be desirable in the short run, for example giving an employee who is overworked a break. But this is only desirable in that giving that employee a break now will mean their productivity will increase in the long run. So the only balance required for the employer is the degree of increase in productivity and the cost required for this increase. It should be noted that employee happiness correlates with productivity. Therefore any interests of the employee will also be of interest for the employer, but to a lesser degree. But discerning the interests of workers is a more complex question. Positive factors of an increase in productivity such as work fulfilment and economic benefits are clear. But a number of other factors depend on the means by which their productivity is increased.

If the employee is working too hard, they may experience the negative effects of burnout or fatigue. If they are spending too much time working this may be at the expense of time spent with friends and family, or on hobbies. It may also be at the expense of time spent on personal projects, which might improve their skills and knowledge, which may lead to a promotion or better job opportunities.

If the employee is more productive as a result of certain frameworks or tools which measure productivity, the positive and negative effects of those aids should be considered.

The process of measuring workers might have a negative effect in and of itself. It may cause the employee to feel like nothing but a cog in the machine, with little differentiation from other workers. They may dislike being objectified, constantly watched, or the feeling of always being tested. These issues can’t be alleviated unless the measuring is either not done, or not known to the worker. This can in part be achieved by avoiding obvious monitoring devices to avoid reminding employees that they’re being monitored. However it should be noted that there are laws in Ireland saying that workers must be informed if they’re being surveilled, and must have access to that data (Citizens Information, 2018).

Each tool for measuring productivity necessarily collects the information of the workers. This information may differ in degree depending on what the tool is measuring. Consider a tool which measures the amount of time an employee spends on certain tasks against a tool which measures the social network of that employee. The first tool has relatively less information which might be personal or otherwise sensitive than the second. Thus the employee will have more of a negative push from the latter than the former. This may be a negative for the worker only where malicious actors are involved, but they may also value their privacy for its own sake. In this case any collection of personal data will be against their interest proportional to its sensitivity, even if that data is held by a manager they like for example. Security measures may be put in place to help prevent the first issue. However, this second issue can never be overcome as long as measurement is being performed. Furthermore, the malicious actor might be the employer, or someone else with access to this secure information. Thus the negative elements of collecting data about employees can be partially alleviated with improved security, but never entirely overcome.

Furthermore, the employee’s knowledge of this information might alter the actions of the employee in positive and negative ways.

It may increase their productivity, but it may also help them to overcome issues they were facing that brought them frustration. Similarly it might help them figure out which areas they enjoy and which they dislike, and recommend certain tasks over others. It might remove the weight of managing their time themselves; taking up a personal assistant role. It might get the employee recognition and reward for their hard work where it previously went unnoticed. However there are also negative consequences.

This knowledge might pressure the employee to act in ways they dislike. For example, consider an organisational network analysis tool which notifies an employee that their social circle is small. In order to improve productivity, they should increase the number of people they communicate with. However, they’re a private person who prefers fewer contacts. They may be pressured, either by higher-ups or by themselves, into acting against their own interests. This problem could potentially be lessened if the employer has knowledge of the personality of the employee. If they knew that an employee was private, they would know that forcing them to increase their circle of communication would reduce their happiness. If this reduction in happiness is greater than the increase in productivity, it wouldn’t be performed, which would be in both the manager and the worker’s interest.

Workers might also have a reduced ability to work less. If their precise degree of workload is known to their manager, that manager will be more likely to know and act in turn when they’re not working hard enough. Therefore there will be a greater pressure to work harder, even at the expense of their own wellbeing.

If knowledge of the relative levels of productivity between co-workers is known to those workers, a number of issues might arise. Firstly, this might further pressure them into acting in service of productivity at the expense of their personal wellbeing. Secondly, this might create an atmosphere of competition among the workers. This may be positive, where employees enjoy a sports-like competition. However, if the competition is taken more seriously, it may disrupt the relationships between workers: distancing themselves from one another and being less open about issues they may be having. Here the workplace is no longer an enjoyable place for the employee, and even if their productivity increases, it has been at the expense of their happiness. There is however a possible solution to this. If the employer knows beforehand or during the time the employees have this information that this approach won’t work or isn’t working, they can decide to keep the data from the employees. It can still be measured and utilised by the employer, but any potential benefits of the employees knowing this information will be lost.

In deciding whether or not to use these tools and approaches to measure worker productivity, the question that must be answered for employers, employees, and everyone else is to what degree this works. The worry is that, driven by a strong desire to improve productivity, employers will continue to use them even when they might not be significantly accurate or useful. This will result in time and money spent with a poor disproportionate payoff. Similarly, regardless of if it works or not, if productivity measurement tools are being used, employees will feel the negative consequences of them. For example even if having data on the social networks of employees is useless, the employees will still have lost privacy in that regard. Employees rarely have much say in the decisions made by their firm, but this may be one complaint they can issue that will always be taken seriously: if measurement doesn’t work well enough, and it costs both me and you, why are we doing it? Both of these factors will impact the opinion of each member of society, and by extension any legislation that might be passed by a representative body. Therefore from all angles, for all parties, whether or not to proceed with and expand productivity measurement at least requires answering this question: does it work?

Well, it’s hard to say. From my research, I’ve found little data regarding the increase in productivity of firms after using any of the tools listed above or any others. PluralSight boasts 295% return on investment following adoption of their tool as found in a study commissioned by them (North, 2017). However I found no such data for the other three tools explored. The issue here is that I can’t know if it’s I just haven’t found this data; I can’t know what I don’t know. It’s possible that this information is just not publicly available. Firms can always track the increase in their own metrics over time after adopting a tool, and they would have no reason to release this information. So there are a number of cases. In the case that the information exists, but is held privately, a number of issues arise. Firstly, it becomes hard for a firm thinking of adopting these tools to determine whether or not to do so. Furthermore it becomes difficult to determine which of these tools to choose from. Secondly and more importantly, it becomes difficult (if not impossible) to gauge whether the field of productivity measurement generally is currently effective. From this; improvements could be found by analysing the effectiveness of each metric; and it could be determined whether or not this field is feasible in the first place. In the case that this information exists and is publicly available, these issues are gone. The effectiveness of these tools (and productivity measurement generally) can be calculated and compared against the costs to determine if the tool should be used, and if productivity should be measured.

References

Cataldo, M. and Herbsleb, J. (2013). ‘Coordination Breakdowns and Their Impact on Development Productivity and Software Failures’, IEEE, 39(3), pp. 343-360.

Citizens Information (2018) *Surveillance in the Workplace*. Available at: <https://www.citizensinformation.ie/en/employment/employment_rights_and_conditions/data_protection_at_work/surveillance_of_electronic_communications_in_the_workplace.html#:~:text=or%20urinal%20areas.-,Monitoring%20you%20without%20you%20knowing,this%20is%20called%20covert%20surveillance.&text=If%20no%20evidence%20is%20found,should%20stop%20the%20covert%20surveillance> (Accessed 20 November 2020).

Cox, M., Shepard, R. and Corey, P. (1981). ‘Influence of an Employee Fitness Program Upon Fitness, Productivity and Absenteeism’, Ergonomics, 28(10), pp. 795-806.

Graziotin, D. and Fagerholm, F. (2019). *Rethinking Productivity in Software Engineering*. 1st edn. Berkeley, CA: Apress.

Harding, B. (2020). ‘CEO-Favourite Code Metrics, by Data Source’, Amplenote, 1 January. Available at: <https://public.amplenote.com/YQ8EcNuhWbMH7oxhqreK27qz> (Accessed 19 November 2020).

Lindquist, J., Sauermann, J. and Zenou, Y. (2015) ‘Network Effects on Worker Productivity’, CEPR Discussion Papers, 10928(October/November), pp. 1-49.

Niemela, R. et al. (2002). ‘The Effect of Air Temperature on Labour Productivity in Call Centres - a Case Study’, Energy and Buildings, 34(8), pp. 759-764.

North, J. (2017). *The Total Economic Impact of PluralSight*, Cambridge, MA: Forrester and commissioned by PluralSight. Available at: <https://www.pluralsight.com/content/dam/pluralsight2/the-hub/whitepapers/TEIofPluralsight.pdf> (Accessed 23 November 2020).

OECD (2016) ‘Defining and Measuring Productivity’*.* Paris: OECD. Available at: <https://www.oecd.org/sdd/productivity-stats/40526851.pdf> [Accessed 17 November 2020].

Shih, Y., Huang, R., Chiang, H. (2012). ‘Background Music: Effects on Attention Performance’, Work, 42(4), pp. 573-578.

Worchel, S. et al. (2011) ‘Social Identity and Individual Productivity Within Groups’, British Journal of Social Psychology, 37(1), pp. 389-413.

Yano, K. et al. (2015). ‘Measuring Happiness Using Wearable Technology’, Hitachi Review, 64(8), pp. 97-104.