In industry the smartness of the employee is often rewarded, as well it should be. Those with the highest levels of expertise are generally a lot more productive than the average employee and in many cases difficult to come by. It is a combination of both this employee's usefulness and rarity that economics tells us sets the expert's salary at an enticingly exorbitant amount . But to what end? Presumably these companies that employ smart employees are themselves in pursuit of smart goals. SMART goals - defined as being Specific, Measurable, Achievable, Realistic, and Time-bound. It is a generally accepted principle that any goal that you set for yourself should follow this SMART structure, having a start, an end and a series of milestones in between at which you can check well defined metrics in order to track your progress towards your eventual goal. Although this is pretty much universally regarded as a good system for tracking your goals, the jury is very much still out on whether a similar definition of SMART should apply to a company's employees. Throughout the course of this essay , I will examine the industry's attempts to design and apply metrics to their employees in order to implement some degree of "measurability" as a system of determining their most valuable assets as employees. I will suggest flaws both in the implementation of the current approach of measuring Software Engineering and the ethics involved in this undertaking. I will also make my own suggestion as to a superior approach to the current implementation in measuring employee efficiency.

It is obviously a good idea , when considering the effectiveness of metrics used to measure software engineering , to first discuss why these metrics are being generated. Although Software Engineers are an essential resource in the vast majority of any large enterprise these days, we are certainly not cheap. The average salary of a Software Engineer working for Amazon in Dublin is €145,000 (Irish Times). If a company is willing to shell out over one hundred grand, year on year , then they want to be sure at a minimum that they are getting value for their investment by way of a productive employee. What value for money means in the context of Software Engineering is of course ambiguous but I will attempt to tease that out a bit more, later in my discussion.

Obviously it is a good idea to promote those employees whose work is exceptional. It is very basic psychology that if you reward a behaviour that you would like to see persist, then this behaviour will persist. This is not only true of dogs and rats and creatures of inferior cognitive ability whose behaviours we would seek to tailor but according to clinical psychologist Jordan Peterson, this is very much the case for humans as well (Get link for best relationship advice, jordan peterson). If you have an employee whose work you are satisfied with, then you should reward that good work as a means of conditioning the employee to continue to produce work of a similar caliber . The worst thing you can do as an employer or a manager is let someone's good work go unnoticed as this will surely lead to that person feeling disenfranchised. Why should they go the extra mile to produce really good work when there is no monetary compensation or even appreciation for their effort?? For this reason it may be desirable for a company to monitor with exacting scrutiny , the impact of each Software Engineer on their projects, in order to most fairly allocate pay rises and promotions. Rewarding good work in this fashion may also incentivise other employees to work harder if they see that their effort is not going to go unnoticed. It is therefore in a companies best interest to identify their most effective employees and compiling data about each employee and using this data to make this decision may seem like an attractive option.

Conversely a company may also desire to produce this data in order to identify their weakest employees. Like so many things in life, I suspect that employee productivity falls into a bell curve with most employees expending the mean amount of effort to do their job. Some employees will work lots of overtime and be more productive than the average employee and some employees will be a lot less productive than the average employee. Perhaps these unproductive employees very much dislike their job, maybe they are ill or have a complicated situation at home or maybe they are just poor Software Engineers incapable of performing at the required level. Whatever the case may be, it is these employees who contribute the least to the company. If it came to a situation where a company had to make some of it's employees redundant, then it would make sense to insure you retain your most productive engineers and shed some of the less productive engineers as they just don't bring as much to the table. A company may also wish to identify these unproductive engineers in order to reach out to them, perhaps even provide them with further training , that they might upskill and become more productive in their work. In either case, ideally a company would like to be able to generate this sort of metric based data on it's engineers and make an informed decision understanding who their most valuable employees are and which of their employees are essentially expendable.

A final reason why a company might want to generate this kind of data about it's employees is to build a statistical model about project productiveness. From this data , you could potentially extrapolate statistics based on the data e.g "The majority of Engineers are most productive between the hours of 11am - 3pm when beginning work at 9:30am", "The optimal time for daily meetings was found to be 25 minutes", "Projects undertaken in python are completed on average 15% quicker than projects undertaken in java". Using this kind of statistical information, a company could potentially reform it's entire structure of operation , saving itself millions in the process. As we see with all big companies that have collected vast stores of useful information , the information is rarely left to sit in a database on some distant server. The fact that this data has the potential to save a company money immediately gives this data monetary value. And what company knowing that it has a source of income at it's fingertips for a product that is easily distributable would simply sit on this knowledge and not seek to capitalise on it? I believe that in measuring the productiveness of it's software engineers, almost any company having compiled this data and used it to streamline it's own processes, will ultimately move to farming this data for financial gain and selling it off to the highest bidder.

So now we understand the motivation behind collecting this data, it may be time to consider exactly what data is being collected and how useful is this data. We live in a world of data, where virtually everything we do is tracked and logged somewhere on the internet, added to a set of big data and analysed at a later point. Similarly for the software engineer, most everything they do from the moment they come to work to when they leave in the evening is being broken down into crude unprocessed data. Professor Barrett even discussed with us a project on kickstarter called ZIE which has grabbed a lot of attention from employers globally ad which can be used to precisely log employee data pertaining to the work they are doing. Using this small device an employee can individually account for every moment they spend working, drinking coffee, even going to the bathroom! All this is achieved by rotating a small desk ornament as the employee moves from task to task. The side of the desk ornament facing upwards corresponds to the task that the employee is currently undertaking. All this data can be logged in an app and exported to a report so that an employee can account for their own productivity on an individual basis.

Although this project will surely produce a lot of data, the data it produces is not specific to the software engineer. That is not to say that there have not been other attempts made to quantify the productivity of the software engineer. In fact there have actually been many. There is an ongoing effort to define metrics which can be applied specifically to the work of a software engineer. For the purpose of this report I will divide these metrics into time based , quantitative based and code based metrics.

In terms of time based metrics, these vary greatly depending on whether a company is implementing an agile or waterfall development approach. If the company is following a waterfall approach, then there will simply be a number of milestones or deadlines to be met throughout the course of the project. If however, the company has adopted the agile approach, then there is a series of other metrics which can be used. Such metrics include:

**Time spent completing subtasks**: All subtasks should be sufficiently modularised that they can be completed in less than four hours. A developer taking longer than this will  cause a stall in the pipeline of production as other developers will be waiting on this work to be completed.

**Meeting Times:** Do meetings start and finish on time? If so then this is a good indication that the project is progressing smoothly, the clients are happy and the team is making efficient progress.

**Defect Lifespan:** How quickly are defects being closed on average after they have been opened. Good software engineers will making many small commits, making incremental changes. If there are a series of large defects that take a long time to close being committed on a regular basis, then this is obviously not efficient software development.

**Up Time:** If you are producing a web based product, then the percentage time that it’s up is an essential statistic in detailing how efficient the deployed code is.

**Time From Code Complete To Deployment:** This statistic details the length of time between an engineer’s code being complete and when it is deployed. This is heavily dependent on how the code fares at the Quality Assurance testing stage. If the engineer has not thoroughly tested their code , then many bugs will be uncovered in their code and it will be passed back to the engineer. This is normal but if the code is continually passed between the QA department and the engineer as more bugs emerge, then it is perhaps an indication that perhaps this engineer is not as thorough as they should be.

Code based metrics are less dependant on the development approach that any particular company has opted to take. They pertain solely to the quantity and quality of the code that a developer has written. Examples of code based metrics are :

**LOC:** a rather crude descriptor of the productivity of a developer based on the sheer number of lines of code that they have both added and deleted from a project. This method of evaluating developer work load simply counts the number of return characters in the developers code and does not take into consideration the syntactic intricacies of a given language or indeed large chunks of white space that a developer might have in their code.

**Percentage code currently active:** A good indication of how good a developers code is, is the percentage of their total coding contribution that is still used in the current deployment. For clarity, you can imagine a situation where one developer contributes ten thousand lines of code but only two thousand lines of this code is still in the current deployment of the project. Contrast this with a developer who has produced two thousand lines of code, one thousand eight hundred of which are still in the project. This might be an indication that the second developer is writing better quality code and introducing fewer bugs into the project.

**Number of bugs introduced :** There is nothing wrong per say with introducing a bug to a project, in fact , most sensible people will argue that this bugs are a necessary byproduct of implementing a complex system. Obviously in a utopian project, developer would never introduce any bugs into the code base and all previously written iterations of the code base would be full compatible with future revisions. This is entirely unrealistic however. Bugs are to be expected. The problem arises when a developer is introducing a large number of bugs to a project, which they then have to go and fix. As a general rule of thumb, if a software engineer is spending more than twenty percent of their time refactoring code to work out bugs that they have introduced, well then it is possible that the engineer is not writing high quality code and is thus not being economical with their time or the company’s money.

The final set of metrics I would like to discuss in this “How Software Engineering is Measured” section of this paper is what I am calling quantitative metrics. These metrics correspond to cardinalities of tangible concepts. Where other metrics discussed above will either define their own units e.g LOC (lines of code) or used predefined scientific units e.g time spent completing subtasks (seconds) , these particular metrics are simply defined by the total amount of themselves at any given time e.g ten features implemented. Other such metrics include:

**QA Kickback:** As discussed before, defective code will be passed back to a developer when the defect is uncovered at the Quality Assurance testing stage. This is what is called QA kickback. As discussed before, the amount of time between a developers initial completion of code vs when it is deployed is a concern. Also a concern however is the number of times the code is being passed between the QA team and the developer. Even if the developer can quickly patch and resubmit their code to the QA team for testing, if the code is still faulty and the QA team have to pass back to the developer a number of times, this is still a further cause of inefficiency on the developers behalf.

**Velocity:** This metric applies specifically to the agile design process . As part of this process the development team will continually set themselves sprints. This is a series of features in which they must implement before their next scrum meeting. Scrum meetings typically take place every two weeks . The number of features which a team has completed is called the velocity of the project. Obviously a high velocity is good in most cases as it means your team is performing efficiently.

**Number of Features Completed On Time:** Very much linked to the velocity of the project, an important factor in measuring the efficiency of a team is taking note of how many features the team has managed to implement ahead of schedule. Client satisfaction is achieved by bringing the client's project to life. This can only be achieved by implementing features which the client can see working. If it is the case that your team is producing all the features on time for the client, it might be worth considering if the team is currently at optimal capacity or if more features could be implemented ahead of the client's deadline.

**Degree of Engagement:** Employers will often try to piece together information about how well their team is engaging with one another. I don't know that there are many people that would refute the correlation between how well a team communicates with one another and the amount of time spent refactoring code due to a miscommunication. As discussed in the "No Silver Bullet" article that we read at the start of term, software is intrinsically complex. One of the greatest costs involved in any software development project is that incurred from miscommunication between team members about the complexity of a given feature. In the past team engagement would be supposed by asking team members to complete regular surveys in which they would have to rate their engagement with their team on a scale from one to ten. This approach I would argue is sufficiently subjective as to be all but meaningless. In recent times, a far more scientific approach has been employed to collect this data. Most companies these days employ a networking app such as Slack, which teams can use to communicate and which can supply employers with valuable statistics about the individual engagement of team members in the group. This is certainly a growing trend as the McKinsey Global Survey reports that "ninety nine percent of executives whose companies employ social technologies (such as slack) report measurable business benefits". So certainly , it is a pretty safe assumption that moving forward, we are going to see more and more companies adopting these social technologies , capable of compiling and organising this data about their users.

So as you can see, there are a vast number of ways in which we can collect data about the workload of a developer and potentially use this information to make any given project more efficient. But is this effective ? How telling are these statistics that companies are investing sizeable sums of money in producing .Is scrutinising the work of these professionals a good thing?? British software developer and well known author Martin Fowler would beg to differ. He once said "I can see why measuring productivity is so seductive. If we could do it then we could assess software much more easily and objectively than we can now. But false measures only make things worse. This is somewhere I think we have to admit our ignorance". This sentiment was once echoed by Bill Gates who scoffed at the idea of using lines of code as a metric to represent developer efficiency; "Measuring software productivity by lines of code is like measuring progress on an airplane by how much it weighs".

Obviously I am not going to be naive enough as to contradict these men, titans in their own right. I would however refute their immediate dismissal of all attempts to measure the output of the software engineer. The  underlying argument here, seems to be that software engineering is analogous to an art form with the engineer embarking on a creative journey, constantly refining their own work to produce an eventual masterpiece. Hence the output of the engineer will be sporadic and non linear as we would expect an artist's to be. I am not convinced that all attempts to quantify the work of the software engineer are futile however and I will present this argument, as well as my criticisms of the metrics described above, using a series of analogies of my own.

I am not sure how familiar you are with the film moneyball. Starring Brad Pitt and Jonah Hill, the basic premise of the film is that Brad Pitt presides over the worst team in the national league baseball tournament. Based on a true story, Pitt in desperation implements a statistical model employed by Hill as an alternative metric for determining the worth of players. Using this model the star players are traded away in favour of far less valuable players. Players unlikely to hit a home run, but very likely to make it to first base. Very controversial but based entirely on this statistical model , the team who had previously been bottom of the league go on to win the competition for the first time in their history.

If nothing else this story captures the power of statistical models in achieving efficiency. The title of the movie is a reference to how the game of baseball was no longer about talent per say. By simply expending money on the right resources, one could win the league using only the power of data science. The field of sport is of course supposed to in some respect emulate the field of battle, the free market in which there exists the conqueror and the conquered. And just as in sport, theoretically we should be able to apply a statistical model in business, a formula by which to win the game. This is why as Fowler suggests, being able to measure the output of a software engineer is extremely desirable. The problem highlighted by Gates and Fowler is that we do not have strike outs and home runs in software engineering.  There is no batting average or any other clearly defined statistic by which to measure the efficiency of the software engineer. Worse than this, in industry it seems there is absolutely no consensus about which statistics are the best ones to use to characterise efficiency with some companies such as Nearshore Systems even employing LOC  as one of their chief metrics of throughput. So unlike Gates and Fowler, I don't object to the concept of measuring a developers output, I think it is a good idea and something we should continue to pursue in the hopes of one day building an effective model. I don't accept that it is "somewhere we have to admit ignorance" as Fowler suggested. I think that is quite a defeatist attitude. I do however take exception to some of the methods by which we currently measure the engineer. My main arguments are as follows:

**Lines of code:** As Billl Gates suggested, an archaic method for measuring the output of a software engineer.  It is commonly accepted that a language that is more highly abstracted from machine code can complete the same function as the low level language with fewer lines of code. For instance , adding together the contents of two memory locations and storing the result at a third memory location can be achieved in one line of code in any high level language such as python or javascript:

```
var a = b+c
```

To implement this same functionality in assembly code (as an example of a low level language would require two load instructions, an add instruction and a store instruction. Has the low level developer achieved a higher level of productivity than the high level developer? No, and yet they have produced four times the number of lines of code. Using lines of code as a metric is not effective in this case as both developers have implemented the same functionality and so have been equally productive. Extending this point out a bit further, using this method to measure efficiency does not encourage conciseness of code. Typically, the more concise a developer can make their code, the better. If I am a developer and I see my code can be shortened, then for the sake of the project I should shorten this code. But if it is going to negatively impact my performance in terms of lines of code contributed to the project, then why would I bother? Using lines of code as a metric for productivity , more than not encouraging good behaviour for developers, I would actually suggest rewards the bad habits of messy developers which will impact the overall efficiency of the project.

**Stat Padding**: Again, here I make another sports analogy as this is where statistics are most widely publicised. This time to the NBA and Oklahoma City who have one player Russell Westbrook who is currently in the news for smashing NBA records. Statistically he is projected to end his career as one of the best players of all time. Across the basketball community however there is an outcry as the statistics produced about him are largely fabricated. That us not to say that they are falsified or in some way “faked”, rather that his teammates play with the specific goal of improving his statistics, allowing him to score points that they could definitely score themselves among other more basketball specific statistics which I will not discuss in too much detail. This is simply human nature. I see it in my own retail job, if you tell someone that their progress is being measured using a specific set of statistics, then they will alter their performance in order to insure that these statistical measures are favorable. The problem is that as with Russell Westbrook and Oklahoma City, when one does this the stats are no longer representative of the performance as the performance has been tailored to suit the stats. From the statistics, it would seem that Westbrook is having a greater impact on his team than even the great Michael Jordan would have had (widely regarded as the best player of all time). In reality though, this is simply not true. Similarly in software engineering, a statistical model could be manipulated by an engineer to make it seem that they are performing at a higher level than a superior engineer. It is only, in my opinion, when engineers are not cognisant of the fact that they are being monitored, that an efficient statistical model of software engineering performance could be produced.

**Not all Productivity Reflected By Code:** In some cases, the impact an employee has inside their team can be seen as a correlation between the output of the team and not the output of that individual employee. Again drawing an analogy from sports, it was noted only in his final season that Manchester United's win rate would increase dramatically when Michael Carrick was playing. Why did it take so long for staticians to notice that United won over 50% more games when Carrick was playing, 100% more goals scored and less goals conceded. Perhaps it was the fact that Carrick's personal stats do not make him look like an above average player. Few assists with fewer goals and not known for his defending. And yet it is still undeniable that his team functioned better when he was involved. The exact same is true of a development team. A developer might not contribute an excessive amount to the code base, on the surface their statistics might reflect those of the average engineer. Even still the communication and leadership that that same developer may bring to the team may increase the team's productivity and efficiency. The value of this engineer to the team would not be identified by the statistics.

**Interpretation Of The Statistics:** A company collecting this much data about their employees will presumably make this information available to the superiors of the employee. I would not have a problem with this data being used as an indicator for the general healthiness and progress of any given project. I would however take exception to this data being used to grade employees and determine which employees are valuable and up for promotion and which employees are disposable and can be made redundant. As discussed above some of these statistical measures are not entirely telling . It is then the prerogative of the employer to place a value on these statistics in determining how "good" or "bad" an engineer is. In the vast majority of these cases, the people making this determination are not staticians. We have already discussed in detail the complexity of trying to measure the output of the engineer. When given this data it is tempting to interpret this data using a univariate analysis. i.e to look at a single statistic e.g number of defects introduced and point to it solely as an indication of the engineers productivity. This is absurd however, it is only by considering a number of these statistics in a multivariate analysis that one can truly make any observation of an employees individual efficiency. I am not convinced that in all cases , a supervisor would go to the bother of putting together such a multivariate analysis but without it, I think collecting this data is just pointless.

**Statistics Stalling The Pipeline of Production**: It is my own experience in a working environment that there can be a certain amount of bureaucracy surrounding who should bear the brunt of statistics that would impact negatively on an employee's reputation. For instance in the company that I was working for during the Summer, it would often be the case that a developer would submit his code for testing to the test team. The test team might test the code and raise a defect against the code if they determine that it is not working. The code will then be passed back to the developer to fix the bug. Occasionally however, the developer would refute the validity of the bug and pass the code back to the test team. The test team would not want to close a defect that they raised as it would negatively impact their statistics and make them look incompetent. I did see on a number of occasions, a back and forth between developers and a test team, where working code would sit in production and not be passed on to the next stage of deployment just because of the bureaucracy surrounding the statistical implications of admitting liability for a defect. I don’t believe this to be an isolated incident, in the same way that an employee might pack their stats as i discussed above to appear better than they are, an employee might also waste time refuting statistics that would negatively impact their reputation. This just slows down the entire pipeline of production and is more of a nuisance than anything else for a client whom you are trying to please.

There are also of course a number of ethical implications to be taken into account when collecting such a large volume of data about your employees.