Measuring Engineering

“The best way to kill software engineering productivity is to measure it” - a predominant theme I found when researching this topic. “Measuring productivity isn't the problem, it's what measuring does to the productivity.”

No one enjoys being measured. Measurement means comparison to peers. Measurement results in either shame or unwanted praise, or neither and a malign mediocrity, unworthy of either side’s attention.

The solution to this for those measured seems to be less apparent systems, less obvious results. For the measurers this would be utopia - no accountability! Take what data they want and do with it whatever they wish. Provided no problematic people, like legal folk, start asking questions.

So that won’t work either. This is a difficult problem to ameliorate for all parties. Does the problem lie in upper management searching for meaning in vast swathes of noise, or is it the software engineer’s preprogrammed aversion and skepticism towards productivity measurement?

Software engineering is a mysterious and magical process where a problem is addressed with a solution dreamed up in 1s and 0s by software engineers who were brought up on explanations full of black boxes and extreme pragmatism. So how can we measure this sorcery?

The measurement of software engineering has been a frustrating topic since the practice was formalised. Measurement is a pillar of the sciences. Without quantitative measurement of one’s results how can procedures be shared, analysed and improved on? Lord Kelvin, Irish-Scottish pioneer in many scientific fields said of measurement:

“When you can measure what you are speaking about, and express it into numbers, you know something about it; but when you cannot measure it, when you cannot express it in numbers, your knowledge is of a meager and unsatisfactory kind: It may be the beginning of knowledge, but you have scarcely in your thoughts advanced to the stage of science.”

In essence, measuring without meaningful data is guessing based off glorified hunches. In fact, measuring something without being able to measure the full scope

of the subject is pointless. Robert D. Austin's book "Measuring and Managing Performance in Organizations" came to the conclusion that unless you are able to measure all aspects of a given activity, say software engineering, the system will be gamed. As there are so many variables, some measurable, others not, in software engineering, Austin came to the conclusion that it cannot be reliably measured, and therefore should be termed evaluation rather than measurement, as it is, at some level, a subjective view of the process of producing software.

Still, just because it's a difficult task does not mean it is not worth the effort. In terms of software engineering, meaningful data is useful in evaluating the work that is being done. Can we be sure it is efficient and not a massive waste of precious time and resources? This is not as much a concern for the engineers as for those paying the engineers for their time and apparently unmeasurable skill. Of course engineers want to work with others who are reliable, who will produce good quality products and who will minimise errors and design flaws. The only way to bring together talented engineers is to have some way to evaluate them.

Software, ideally, should deliver requirements as laid out in an initial document defining the scope of the piece of software. It would ideally take as much money and time as estimated before beginning. The design should be fluid and modular making aspects reusable in future projects. It should be easily maintainable with good documentation so that not solely the original engineering team, or worse, one person on that team, are able to maintain and provide fixes to the system.

In reality, the most you can hope for is that the project delivers the functionality requested by the stakeholder. There is conflicting reports but around 70% of projects fail - they come in over time, over budget, with stripped down functionality, overly buggy or rejected by the users. However, some recent surveys have started reporting improvements in this regard, some citing lowering failure rates of 55%. These are attributed to an evolution on the project management side of the development process:

"organizations are becoming more mature with project management, and are focusing on benefits maturation and realization, instead of solely on cost, time and resources. In other words, there's less focus on the means by which a project is deemed successful and more on the ends: does the project deliver the business benefits promised?"[1]

Software engineering has historically been measured through different software metrics. 1974 saw the first measure introduced; lines of code (loc). The average

[1] "IT project success rates finally improving | CIO." 27 Feb. 2017, https://www.cio.com/article/3174516/it-project-success-rates-finally-improving.html

code rates were measured per month. This metric was used as an overview of general code production rates and comparing month to month could give some useful, if limited, insight[2]. Lines of code is not very practical nowadays. Cleaning up an application, getting it to run six times faster while removing 2000 lines of code would probably result in a quick firing if lines of code were what your performance reviews were based on. When there was one predominant language per company loc may have been a passable metric but now the process is more complicated and there is a requirement to write efficient yet readable code.

The Halstead complexity measures followed in 1977. These expressed programmer's effort, time to program and number of bugs as functions using only the number of (distinct) operators and (distinct) operands. They are strong indicators of code complexity and therefore how maintainable a piece of software is.

Function points came next and are still used to today. These were specific functionalities derived from the user's functional requirements and categorised into one of five types: outputs, inquiries, inputs, internal files, and external interfaces. Each of the functional requirements maps to an end-user function like user input or user query. These are similar to user stories in the Scrum methodology. Function Point Analysis quantifies the functions used within software. The quality of the code is based on comparing the initial requirements to these function points in the release.

Modern times has seen a shift from solely analysing the code to more of an all encompassing view of the software engineering process. There has been a shift from waterfall methodologies to more agile and iterative methods of development. Big data technology available today looks predictably to be incorporating itself into the set of tools available to employers to assess employees.

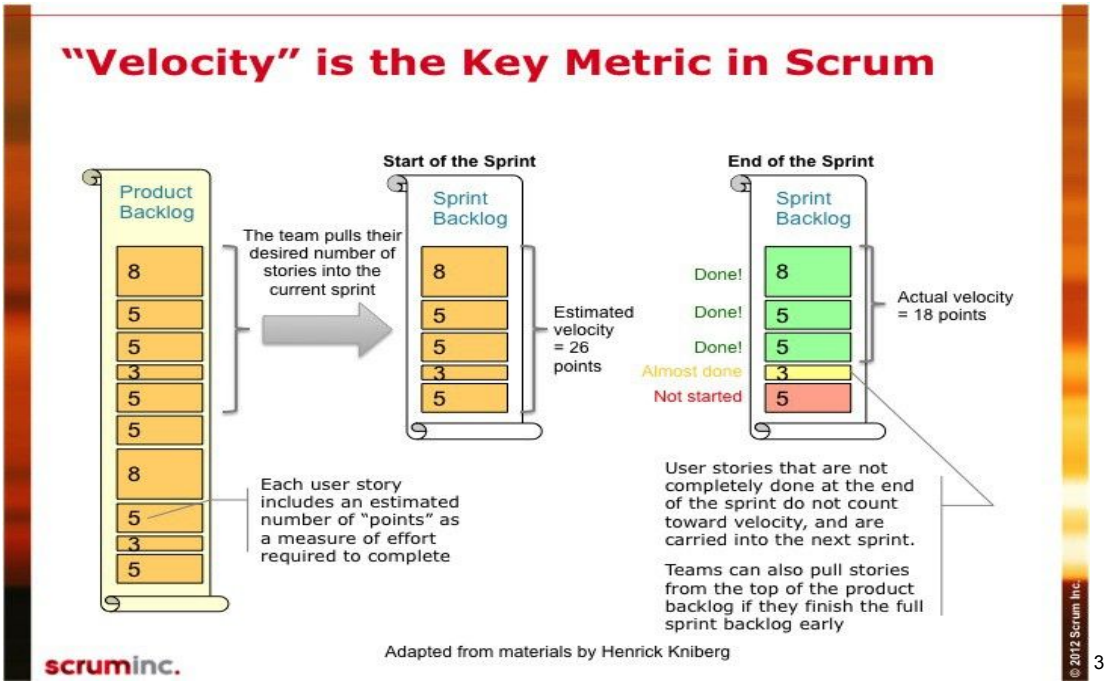
Scrum is a modern form of agile methodology which is often used within software engineering teams. It involves gathering all the functional requirements as "user stories" and then working in short cycles called "sprints" prioritising the most urgent tasks. Scrum is all about constant communication so teams meet daily to stay on track.

The key metric in Scrum is a team's sprint velocity and shows how much work a team can get through in one sprint cycle. A team's sprint velocity should tend upwards by 10% with each cycle as they adapt and make small changes with each iteration. Used in conjunction with Sprint Burndown Charts, problems within teams

---

[2] "science such as physics and some of the less - Software and ...." https://wwwbroy.in.tum.de/lehre/vorlesungen/vse/WS2004/zuseh_metrics_history.pdf

can be spotted and addressed quickly. This is a big advantage to agile methods in general as opposed to slower development methods like waterfall.



Velocity description from Scrum Inc.

However different teams have wildly different purposes so comparing sprint velocities between teams is not a good indicator of relative performance. Tempo and Stability are two more reliable metrics to evaluate performance.

Tempo is the rate at which changes are made to the codebase. How long does it take for commits to go live, and how often does a team deploy code? A solid routine of committing helps keep the team on track.

Stability is what happens post deploy. How quickly can things get back up after a service incident, and how often does a change to the code require a remedial change after the fact.

Notice these are more results based than purely output based. They look at the context rather than some meaningless and naive statistics about the actual code. This gives a more complete view of the code.

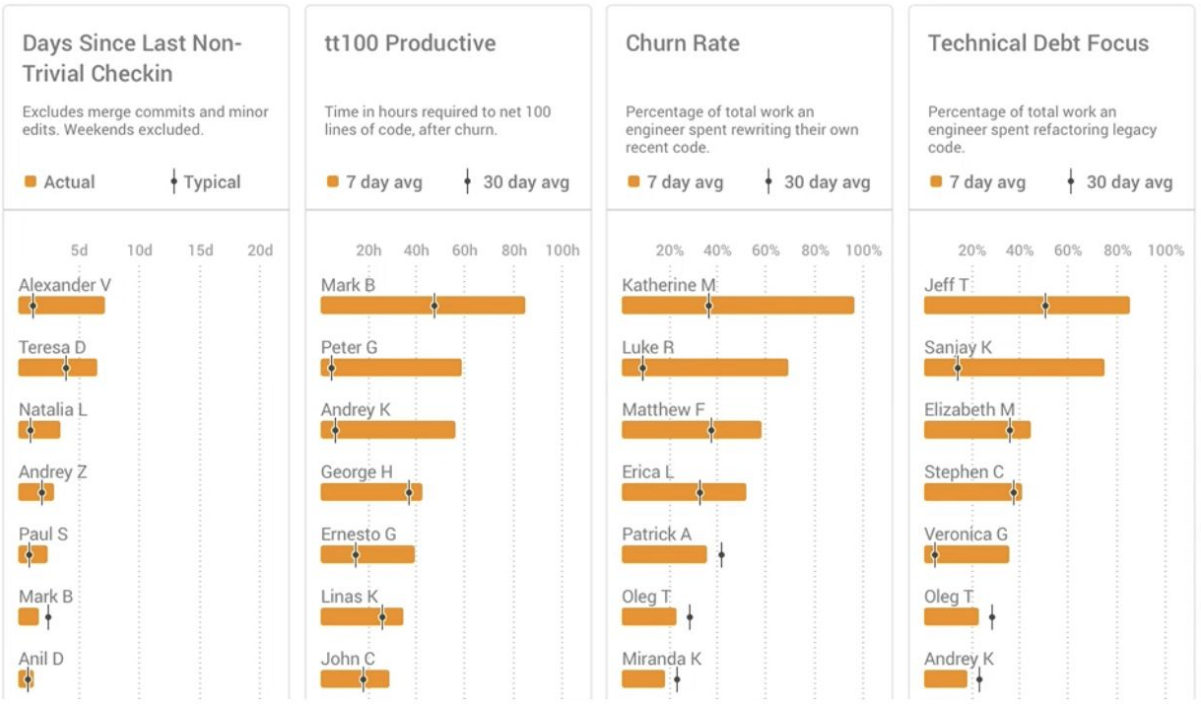
GitPrime and Velocity are two engineering intelligence tools which analyse Git repositories and give insights into code quality. Churn or Rework are metrics that measure how much of what was written previously gets changed. As with basically all the metrics, the results must be interpreted correctly. An especially large churn

[3] "Velocity - Scrum Inc." https://www.scruminc.com/velocity/

rate does not necessarily mean you have a team to fire, but values outside the norm can signal potential disaster, or an unusual bug was caught and reason to celebrate.

Impact is a code analysis metric where changes are ranked by their complexity. Changing a variable name will be simple and optimising an algorithm will be ranked as trickier. Semmle Inc published a report using a similar code analysis tool to measure the quality of repositories which I will explore later on.

Each of these tools have great visualisations of their data and can provide otherwise difficult to acquire insights into team performance, including Scrum methodologies.



Spot Check in GitPrime.

These are some of the code focused ways of measuring software engineering but software engineering is generally a group task carried out in a workplace where the fact that humans' actions cannot be predicted like a machine's can, and inter-team relationships play a big factor. When I mentioned the difference between measurement and evaluation this is one of the factors which cannot be fully measured.

However, no company will ever turn down knowing more about their employees. It's not like Facebook would happily mine their user base for unimaginable amounts of data but let their employees escape.

Behavioural data is another side to the story. If one team's Scrum velocity suddenly starts to dip but they are all good programmers and excellent communicators, well then what do you do? The answer seems is to record all your employees conversations, use an accelerometer to see if they lean in and "engage" in these conversations and track their movements using bluetooth and GPS.[4]

The ethics of it all will come later but you only have to look at the value of our data on the market these days, and not just from the usual suspects, to get an idea of how much this data fuels pivotal decisions. For example, not too long ago Ford found itself in a slump. The industry was changing, cars were going electric, the planet was becoming a priority, so what does Ford do? Maybe use all that engineering power to focus on going green, create some more efficient hybrid engines, increase supply and lower the cost? Needless to say they didn't. Instead they looked at the cash machines of Facebook and Google and began harvesting data from their users and selling it on to offset any downturn they were facing.

The example used in the video cited shows a European bank which trialed Humanyze, a company which tries to "understand how teams are actually working" through a hi-tech badge full of sensors. They saw three main trends in their employee communication graphs; well connected and well functioning branches, split graphs and graphs with outliers, the latter two not functioning as well as branch one. Split branches had two floors and outlier branches were too tight knit for new employees to penetrate.

Humanyze Presentation.

[4] "Using analytics to measure interactions in the ... - YouTube." 10 Nov. 2014, https://www.youtube.com/watch?v=XojhyhoRI7I.

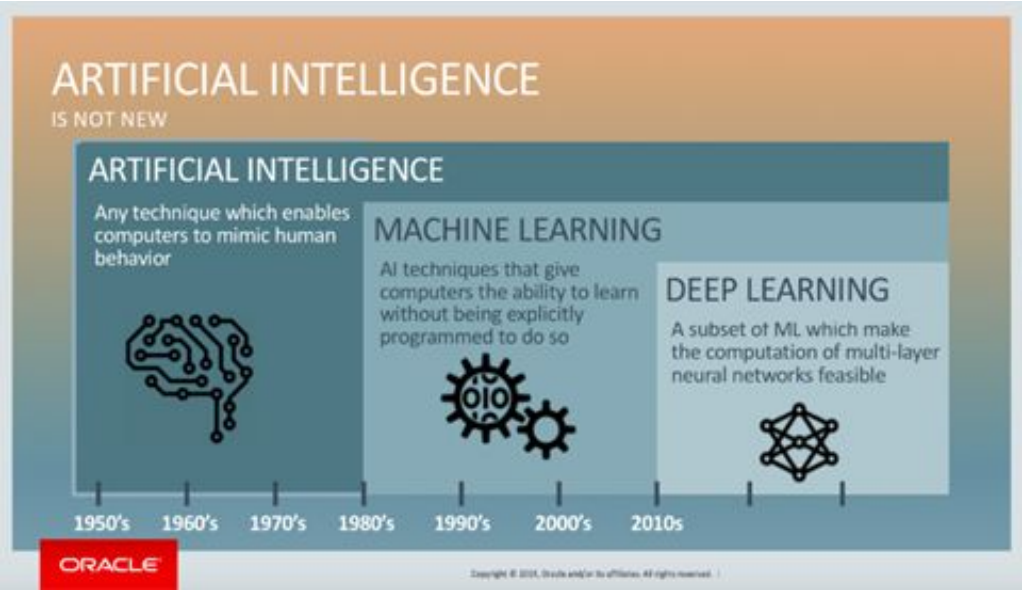
Fixing these issues led to 11% increase in profits. Huge money for a multi-national bank, all from some new wave behavioral measurement. It is obvious that this, in conjunction with current software measuring practices, could yield potentially game changing rewards.

Even with the most sophisticated techniques and although the process is already complicated enough, productivity trends are not as simple as making a change, observing the results, and evaluating your success. Back in 2003 economists were only beginning to see increases in business from the computational investments back in the 90s. Improvements lag investments.[5] So your new measurement technique may take some time to show results. Or it may not show any at all.

With the advances in AI and Machine Learning (ML), and the never ending data we all produce surely there must be some way to feed everything through a computer and figure out who the strongest programmer is. Not so simple. AI and ML are very good at taking things we are already fairly good at and bringing them to the next level by intense trial and error, as seen with AlphaGo and AlphaStar in Go and Starcraft. The current methodologies we use for evaluating software engineers are still subjective, or based on small data sets - two things AI and ML don't work well with.

The thing about AI and ML is that they have been around since the 50's and 60's respectively. AI is a computer program which makes decisions based on some heuristic function. ML is a fancier subset of AI that can figure things out from data and deliver AI applications. "ML is the science of getting computers to act without being explicitly programmed" - from Stanford University's "What Is Machine Learning?" course. Deep Learning (DL) is then ML based on multi layered neural networks.

[5] "CannotMeasureProductivity - Martin Fowler." 29 Aug. 2003, https://martinfowler.com/bliki/CannotMeasureProductivity.html.



[6]

AI is not new.

These algorithmic approaches must be useful in some way as we now have vast amounts of data which is produced during the software engineering process. But is it possible to train some application by feeding it in all the good code we can find, just as we would train it to recognise a cat?

In the current landscape, where exciting headlines and clicks are the top priority, AI is hyped up to the point where the public has a view of it that researchers have no idea how to build. Therefore the limits of AI are much lower than the general populace think. That is not to say that AI is useless, far from it. There are some tentative steps being made into the field by some prominent players.

Granted, intelligent AI assistants are pipedreams and recognising RGB patterns in the near endless pictures of cats from the internet might be a little easier than measuring complex processes but AI can be used to make important resource decisions in companies if we choose the right metrics and make enough data available.

The first forays into measuring software development with ML are being made. Semmle Inc is an internationally based code analysis platform who were bought by Github earlier this year. They published a study “Measuring software development productivity: a machine learning approach”[7] based on their own code analysis engine.

---

[6] "What's the Difference Between AI, Machine Learning, and ...." 11 Jul. 2018, https://blogs.oracle.com/bigdata/difference-ai-machine-learning-deep-learning

[7] "Measuring software development productivity: a ... - Semmle." 18 Jul. 2018, https://semmle.com/assets/papers/measuring-software-development.pdf.

Semmle Inc's LGTM (Looks Good To Me) analysis engine was the massive dataset used in their study. Featuring 10 million commits from 300,000 developers to 56,000 open source repositories it would be hard to find a better starting point. They used their own object oriented language, QL, specialised in querying complex data encoded in relational data models. It "treats code as data", so perfect for this task. I'll give an overview of their process here.

They set out to measure the quantity and quality of the code commits made to the repositories in their LGTM database. LGTM analyses all its code by throwing hundred of QL queries at it looking for all different kinds of flaws. An example in the Linux repository is shown below. LGTM therefore provides supreme code quality data for further processing.

LGTM code analyser.

Two coding changes take the same "amount of output" if they take the same labour time to produce. Labour time cannot simply be the time between commits. If the max time between commit was 15 minutes it could be feasible, but days can go by between user commits, and it has to be assumes they are not coding the entire time.

Here's where the first neural network comes in. It is a Hidden Markov Model meaning it predicts some hidden variables based on some known variables, eg what the weather is (hidden variable) based on people's clothing (known variable).

Based on a committers commit history they estimate how likely they are coding for each previous minute. This is then used to estimate the time spent coding between commits. Basically, short intervals of less than an hour indicate more non stop coding hours and longer intervals of 2-5 hours indicate more breaks. This turns out to be a flexible metric which can differentiate night coders, weekend coders or nine-to-five coders.

These coding time models are then associated with the actual code changes. A model is then trained so that when it is given a new piece of code it can estimate the

time it should take to produce it. This seems to be a good analysis, the model would pick up on projects which have dependencies elsewhere and more complicated projects. Unlike some naive metrics, lines removed is as valid a metric as lines added and is given equal importance as the removal of lines also requires coding effort.

They do note that code change vs coding time is a very noisy metric with unpredictable work interruptions or just plain skill differences from programmer to programmer having severe effects on averages.

LGTM can visualise the average code quality in a repository and ranks the repo from F to A+ based on a reference set of quality code.

Repo's average code quality.

While this was an interesting study and showed how AI and ML can be applied when given a large dataset and some nice models. It still only provides a broad average. But if a team is working from a repository, the culprit of non obvious shoddy code could definitely be found by finding the dips in quality and linking it with the committer, or by comparing the model's predicted time to code with the actual time to code.

It is also unclear if ML or DL *can* go much further or if we are hitting the limits already. As mentioned previously, these forms of programs were thought of before they were possible to compute due to the lack of data, so many barriers had already been theoretically reached before anything had really happened. It is also the question of "Why in the world would I solve the Travelling Salesman Problem with a neural network when a search algorithm will be much more effective, scalable, and

economical?"[8]. Overcomplicating things just to include "AI" in your marketing may generate more hype but may not be helpful.

These types of experiments will only improve with each iteration and as more clever metrics are thought up. But how far will people have to dig to find incorruptible ways of measuring the immeasurable, and will they all be ethically sound?

What is the whole purpose of algorithmic approaches toward measuring software engineering? It could be said that it is an effort to boost productivity, increase efficiency, get the best job done for the client and for the company. All very pragmatic and above board - so if you, the employee, could give your consent and sign here, we'll go right ahead and get to work on streamlining your work environment.

It's a well worn road but constant surveillance used to be a thing of dystopian fiction. In a wave of early 90s technological optimism, some thought up fantasies of how this marvel could be misused. In 1984, it was in your face. Big Brother was feared, always vigilant. It was a frightening life to imagine living. A Brave New World depicted a possibly more realistic future in which the world has been coerced with more subtlety into a seemingly perfect life. Feeling down? Take a hit of soma, it's not that bad.

These were dystopias, worst case scenarios where large organisations were untouchable, integrated into society, where one cannot simply opt out. It feels cliche to draw these comparisons as anyone who has read the books has found similarities within them, no matter the time period they were reading in, but it feels more vivid and more futuristic than ever now.

I respect Ben Waber, Founder and CEO of Humanyze, for not listening to himself as he delivered his keynote presentation on his product. He threw in jokes and made some of the crowd laugh as he wore his hi-tech surveillance device - hopefully his microphone picked up sufficiently joyful vocal tones - imploring others to track their employees' conversations and movements in the name of workplace behavioral analytics.

Managers currently take up this role. Good ones have a feel for their team, who is socially adept to break news or offer support to those struggling. They know who is most suitable to certain tasks, who should not be pushed and who will take on extra

[8] "Is Deep Learning Already Hitting its Limitations? - Towards ...." https://towardsdatascience.com/is-deep-learning-already-hitting-its-limitations-c81826082ac3.

work lying down. This is an attempt to empirically measure team members, but I go back to my opening paragraph, “Measuring productivity isn't the problem, it's what measuring does to the productivity.” I’m not sure my conversations would be very natural with a microphone stuck to me 8 hours a day. I wager I would not last too long in that particular job. Dressing up employee surveillance with fancy tech does not lessen the ethical quagmire it creates.

Shoshana Zuboff released a book in the past few weeks titled “The Age of Surveillance Capitalism: The Fight for a Human Future at the New Frontier of Power”. She tackles the rogue economic logic the tech superpowers of today follow. The Facebooks and Googles of today have set examples where gathering our data has become the norm, an expected price to pay for the use of their services. They have convinced us that their practices derive from the inevitable outgrowth of digital tech. If we want to ride this technological wave we’ve got to go with them. If we want to avail of one of humanity’s greatest achievements, there’s a seemingly invisible but omnipresent price to pay.

GDPR regulations came in last year. Automated individual decision-making is where these companies, who provide a service in return for information mining, find their edge on the competition. They generate their margins over the competition by providing predictive data which is then sold onto those who use it to craft, not what we want but what we will buy. We have been brought up like this so it’s normal but still a shame that the shady practice seems to be spreading into other sectors such as socially tracking software teams.

Facebook had no sign up fee, so maybe it was a fair swap. But what if the company mining you was also paying you, what lengths would they go to to find out if there was better value on the market? What if it was some faceless robot with a cheery tone of voice that gave you passive aggressive notes on your commits, told you who to ask for advice, or made the decision for you to be let go?

GDPR does not protect against all the risks that come with algorithmic decision-making[9]. Where people are directly affected, companies need to be very careful how they go about implementing such systems. Just taking a firing as an extreme example, proper cause is needed. As there are virtually no previous AI cases in this regard, litigation could get messy if the employer is reduced to claiming the computer told us to do it. [10]

---

[9] "Monitoring algorithmic systems will need more than the EU's ...." 24 Jan. 2019, https://ethicsofalgorithms.org/2019/01/24/monitoring-algorithmic-systems-will-need-more-than-the-eus-gdpr/.

[10] "Should you use AI to make decisions about your software team?." https://techbeacon.com/devops/should-you-use-ai-make-decisions-about-your-software-team

Sports science has progressed unimaginably in the past twenty years. When once shovelling down plates of pasta passed as adequate preparation and “football men” were the only ones qualified to analyse football, it was a simpler time. Now players are tracked in real time with accelerometers and through GPS. The level of sports people has improved, the practicality of statistical analytics is unquestioned by anyone learning the game now. Sports people are lucky to be operating in a field with clear boundaries, both on the pitch and in writing, such that it is feasible to measure every aspect of the activity and point to the stats when making decisions, Moneyball being the obvious example, favouring the older, cheaper, less obviously talented baseball players and winning big.

The problem arises in that software engineering cannot be measured so completely. The next social step is being made in an attempt to remedy this. Software engineers are not examined on their physique thankfully but do not get to leave their work at home as much. Commits you make from home on a Friday night to keep your tt100 rate up are analysed just as critically as any other.

Using these ML models to pick the best team for a project is fantastic and would be lead to happier employees and employers as, in theory, everyone would be working on some aspect that they are good at. But, if companies begin using the data for punitive or disciplinary reasons, it becomes a much more complicated matter.

[There is also the matter of ensuring different types of team members are graded fairly. When we were introduced to team work in first and second year -> belbin….]

I believe this is a dangerous path to go down. I do not question the potential rewards that could become of sophisticated measurement but, as discussed, software engineering is such a complicated and intricate process which combines numerous facets of software development and team structure that, to be useful and not create dysfunction, virtually all have to be measured. This conflicts not just with workers’ but with human beings’ fundamental rights to privacy.

It seems like a Catch-22. The current system I see as feasible revolves around mainly coding data which gives only rough averages and estimates to work off. I might feel hard done by if any major decisions get made based off these metrics. On the other hand to feel comfortable having decisions made mostly autonomously, the model needed would be ultra sophisticated, fed on data I would not be comfortable giving away.

We may be far from adequate measures of software engineering but that does not mean efforts should not be made. I think it is an interesting area and breakthroughs

could be industry changing. Although there can be no accountability if there is no transparency which is a dangerous task in the current climate.