# Measuring Engineering - a report

To deliver a report that considers the ways in which the software engineering process can be measured and assesed in terms of measurable data, an overview of the computationial platforms available to perform this work, the algorithmic approachs availabe, and the ethics concerns surrounding this kind of analytics. Reading list available here.

1. **What data ?**
2. **Where to compute?**
3. **What algorithms?**
4. **Ethics**

# Measurable Data

What data can one gather that truly measures a software engineer’s output? How can a highly educated indivdual be analysed when the majority of their work is done in the inner thought processes of their head? A software engineers aim is not to produce the most lines of code, or get the job done the fastest but to provide the best solution possible to the problem at hand. How can you measure this?

An answer to the question “How should a software engineer's productivity be measured?” that reallly struck me was Ori Shalev’s on the website Quora. He replied:

*“It can't be measured, because what you refer to as “productivity” presumes a commonly known path of progress for a task, whereas true greatness is in finding the best path - not following an existing one in faster pace.*

*You can measure drivers on how efficiently they use fuel when driving from point A to point B. A truly great driver could find shortcuts - such as boarding a ferry to cross a bay. That sounds very uncommon, but in software engineering it happens all the time. You would never know if an engineer that you don’t have on your team would solve the same problem as yours in 20% of the effort and 2x the robustness. Measuring productivity would be a distraction from focusing on getting the most value out of your team.”* *(Anon, 2017).*

## **Tried, Tested and Failed Measures:**

Many who attempt to analyse a software engineer’s productivity forget that these people are at the forefront of innovation. They are artists. If one was to measure an artist such as Leonardo Da Vinci’s productivity, how would you go about it? The number of brush strokes a day? Rebecca Elfast is a Swedish painter who is renowned for the her use of as few brush strokes as possible *(Emptyeasel.com., 2017).* Does that mean that if I layered multiple coats of paint over a canvas my art is more productive, or better than hers? Certainly not! The same applies to software engineering, the old standard of using the number of lines of code per day cannot be used any more as one line of a high level language could take 3 days to write and debug, simillar to how one brush stroke could take a whole day to paint.

Another reason why past attempts of measuring output through ways such as lines of code, bugs fixed, stories closed, passing tests failed is because engineers are smart. Clever engineers can easily manipulate the results to make it appear that they are doing more than they actually are. A salesman cannot easily lie about how many sales he made that day as the cash register will present the true value, where as it is easy to space code over a number of line, rather than condensing it down into one or two. Why use a loop when you are being measured on the number of lines of code you produce? If you are measured on whether or not your code passes it’s test, why would you spend time writing code to best solve the task when you could simply write code that passes the tests. There’s a quote about this commonly attributed to Bill Gates:

“Measuring software productivity by lines of code is like measuring progress on an airplane by how much it weighs.” (Wiki.c2.com, 2017)

**Hours Worked**

This is one of the most obvious ones: if you worked 10 hours instead of 8 hours, you should get 125% of the work done. That's just maths. Time and time again, you'll see studies proving that this just does not work for anyone. In fact, working more hours is a great way to decrease productivity.

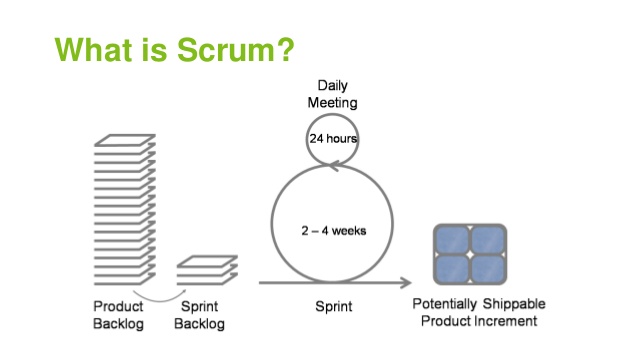
The Relationship Between Hours Worked and Productivity *(Cs.stanford.edu, 2017)*

Henry Ford Drops Hours, Increases Productivity *(week, 2017)*

Stop Working More Than 40 Hours Per Week *(Inc.com, 2017)*

Time and time again, we see proof that more than 40 hours necessarily leads to a drop of productivity, even for assembly line workers. Yet, this pervasive attitude of 8-6 being a minimum workday continues to chug along.

## **Sprints:**

An alternative form of data that one can examine could be the indivdual velocity of a software engineer, the time it takes an engineer to complete a sprint, if your team follows SCRUM as their software development methodology. SCRUM is a form of agile software development, in which the SCRUM team works together to complete the set of tasks they have collectively committed to completing within a sprint.

As described in the Scrum Guide, a Sprint is a time period of one month or less, during which a “Done”, useable, and potentially releasable product is created (Scrumguides.org, 2017). Sprints work best if they have consistent durations throughout the development effort. A sprint backlog is created by the SCRUM team for each sprint and a daily SCRUM, which is a brief team meeting helps set the context for each days work. A new Sprint starts immediately after the conclusion of the previous Sprint.Scrum Sprint is part of the Empirical Processpotentially releasable product Increment is created.

So if you were following this software development methodology, you should be able to estimate to a respectable level of accuracy what's the individual velocity of each software engineer after a few sprint cycles. You could use the individual’s velocity or the team’s velocity as a productivity metric.

For more information on sprints and SCRUM see https://www.scrumguides.org/

## **Using Net Promoter Score to measure employee satisfaction**

As previously stated, there is no one piece of data that one can gather that will acurately measure the productivity of a software engineer (or a software engineering team) as there is no concrete way to measure the output of a software engineer. Without this concrete measure of output, firms can’t easily calculate whether you’re a more or less productive engineer than you were last year, or whether your team is more productive than it was last year. They can’t decide whether to change programming languages, whether to adopt “agile” methodology or reject it, or whether to change the way they interview candidates or select new employees.

Intuitively, such variations in productivity must exist and Refin discovered that when they surveyed engineers regarding what they like/dislike about their jobs, the aspects they complained about appeared to correspond to something regarding their productivity. The employees commented on long compile times, having too many meetings, and a noisy work environment. They also requested another big monitor, clearer specifications, or more time to pay down technical debt. (Code Red, 2017)

Across all industries, satisfaction at work comes down to feelings of autonomy (directing our work), mastery (accomplishment and self-improvement), and purpose (working on something more important than yourself). Unproductivity is the opposite of mastery. It attacks job satisfaction directly. But, unlike productivity, firms can measure job satisfaction.

Redfin is both a real-estate brokerage and a software company and uses Net Promoter Score (NPS) as a standard metric for customer satisfaction. It also uses NPS as a measure of employee satisfaction by asking their employees the simple question: “How likely are you to recommend working at Redfin?”

Redfin admits that they cannot prove NPS is the best measure of engineering productivity, or even a good measure as that would require them to be able to measure engineering productivity to compare them, and as previously states, this is impossible. When they focused on their engineer’s top complaints, things that they said were driving down their productivity, they found that the satisfaction of their engineers rose.

Redfin also advise that just because you’re measuring employee NPS, that doesn’t mean you stop recording other metrics, such as bug counts, user retention, or conversions. That data still matters and can but used to create a more comprehensive estimate of employee productivity. It found that NPS was the best “simple” measure but that a combination of measures can be better.

For more information on NPS as a measure of employee satisfaction can be found at Net Promoter System’s website - <http://www.netpromotersystem.com/about/employee-engagement.aspx>

# Computational Platforms:

A computing platform is the environment in which a piece of software is executed. It may be the hardware or the operating system, even a web browser or other underlying software, as long as the program code is executed in it. For the purpose of this report, I will discuss the past and present computational platformsused to mearure a software engineer’s performance. There are many different firms that offer data analyitics, the website clutch.co lists 209 firms in the “Best Big Data Analytics Companies – 2017 Reviews” but few of these specialise in software engineering.

## **Personal Software Process(PSP):**

In 1989 Watts Humphrey authored the influential book, ‘Managing the Software Process’, in which he developed concrete methods for managing software development and maintenance. These methods, now commonly practiced in industry, provide programmers and managers with specific steps they can take to evaluate and improve their software capabilities. In 1995 he penned an new ground-breaking text, ‘A Discipline for Software Engineering’, in which he scales those methods down to a personal level, helping software engineers develop the skills and habits needed to plan, track, and analyze large, complex projects. He adapted organizational-level software measurement and analysis techniques to the individual developer.

In his text he presents concepts and methods for a disciplined software engineering process, he scales down industrial practices for planning, tracking, analysis, and defect management to fit the needs of small-scale program development and shows how small project disciplines provide a solid base for larger projects. These techniques are called the Personal Software Process (PSP). Such techniques aimed to automate the collection of human data which would, in turn, eliminate human error and human effort.

Humphrey’s version of the PSP uses simple spreadsheets, manual data collection, and manual analysis. Collecting and managing this data takes substantial effort. Interestingly, Humphrey actively embraced the manual nature of the PSP:

*“It would be nice to have a tool to automatically gather the PSP data. Because judgement is involved in most personal process data, no such tool exists or is likely in the near future”.*

More fundamentally, Humphrey viewed his predefined PSP processes as a bootstrapping method. In the book, he exhorts developers to modify the forms and procedures he presents to address specific circumstances and needs.

## **LEAP:**

After substantial research was conducted into Humphrey’s PSP process, it was concluded that the manual nature created problems for significant data quality problems. To address this problem, Leap toolkit was developed. Leap stands for Lightweight, Empirical, Anti- measurement dysfunction and portable software process measurement. It attempts to overcome the data quality problems associated with PSP by automating and normalizing data analysis.

According to the Univercity of Hawaii’s website, Project LEAP resulted from a recognition that many software process improvement initiatives suffer from one or more of the following problems:

* Heavyweight development process constraints. For example, many process improvement initiatives require adherence to strict documentation, audit, and development phase constraints.
* Measurement dysfunction. The use of process metrics for employee performance evaluation can lead to “dysfunctional” behavior which skews the metric in the desired direction while compromising overall organizational performance.
* Organization-level analysis and improvement. Typical process measurements aggregate data collected from multiple projects and organizations. Such data takes time to accumulate, analyze, and produce meaningful process improvements.
* Manual data gathering. Measurement may involve time-consuming clerical overhead that lowers the quality of the data and produces resistance to its collection.

The goal of Project LEAP is to produce tools and techniques to support software process improvement for individual software engineers that satisfy the LEAP constraints:

* **Light-weight:** LEAP support must be “light-weight”. It must be easy to learn, easy to integrate with existing methods and tools, and above all, not impose significant new overhead on the developer unless that investment of overhead will provide a direct return-on-investment to that same developer.
* **Empirical:** LEAP support should be quantitative as well as qualitative. Software developer improvement should be able to be shown through measurements of effort, defects, size, and so forth.
* **Automated:** Light-weight support for empirically-based developer improvement virtually demands some form of automation. On the other hand, automation does not by itself guarantee light-weight processes or meaningful empirical evidence of improvement.
* **Portable:** As a developer-oriented approach, Project LEAP recognizes that any long-term improvement mechanism must accommodate the fact that software developers change jobs and companies on a regular basis. Useful support cannot be locked into a particular organization such that the developer must “give up” the data and tools when they leave the organization. Rather, LEAP support will be a kind of “personal information assistant” for their software engineering skill set. (Csdl.ics.hawaii.edu, 2017)

Although the developer still manually enters most data, the toolkit automates subsequent PSP analyses and, in some cases, provides analyses (such as various forms of regression) that the PSP doesn’t provide. It attempts to avoid measurement dysfunction by enabling developers to control their data files. It maintains data about only the individual developer’s activities and doesn’t reference developers’ names in the data files. It creates a repository of personal process data that developers can keep with them as they move from project to project and organization to organization. By introducing automation, the Leap toolkit makes certain analytics easy to collect but others increasingly difficult. All data collected can then be analyzed by Hackystat, a data collection tool which was developed at the University of Hawaii.

Hackystat is an open source framework for collection, analysis, visualization, interpretation, annotation, and dissemination of software development process and product data (Csdl.ics.hawaii.edu, 2017). It implements a service oriented architecture in which sensors attached to development tools gather process and product data and send it to a server, which other services can query to build higher-level analyses. Hackystat includes four important design features. The first is both client and server-side data collection. Modern software development typically includes individual developers’ activities on their local workstation as well as server- or cloud-based activities. The second feature is unobtrusive data collection. For developers, one of the most frustrating aspects of manual data collection is the loop of doing some work and then interrupting it to record what they worked on. An important requirement for Hackystat was to make data collection as unobtrusive as possible. Users shouldn’t notice that data is being collected, and the system shouldn’t make assumptions about network availability. The third feature is fine-grained data collection. By instrumenting client-side tools, Hackystat can collect data on a minute-by-minute or even second-by second basis. For example, Hackystat supports a measurement called buffer transition—collecting a data instance each time the developer changes the active buffer from one file to another. Hackystat can also track a developer as he/she edits a method, constructs a test case for that method, and invokes the test, yielding insight into real-world test-driven development. The fourth feature is both personal and group-based development. Besides collecting their personal development data, developers can define projects and shared artifacts to represent group work. Hackystat can track the interplay among developers when, for example, they edit the same file.

## **Semmle:**

The field of measuring software engineering created a gap in the market which was soon filled by many diverse and interesting firms. All of these companies provide different platforms with various different features which allow software engineers to analyse various relevant data in a user-friendly, graphical way. Some of the most well-known companies that supply software which collects and analyses measurable data are as follows: Semmle, Gitcop, Gitcolony, Codebeat, Teamscale, Black Duck, Codebrag, Code Climate and Phabricator.

According to Semmle’s website

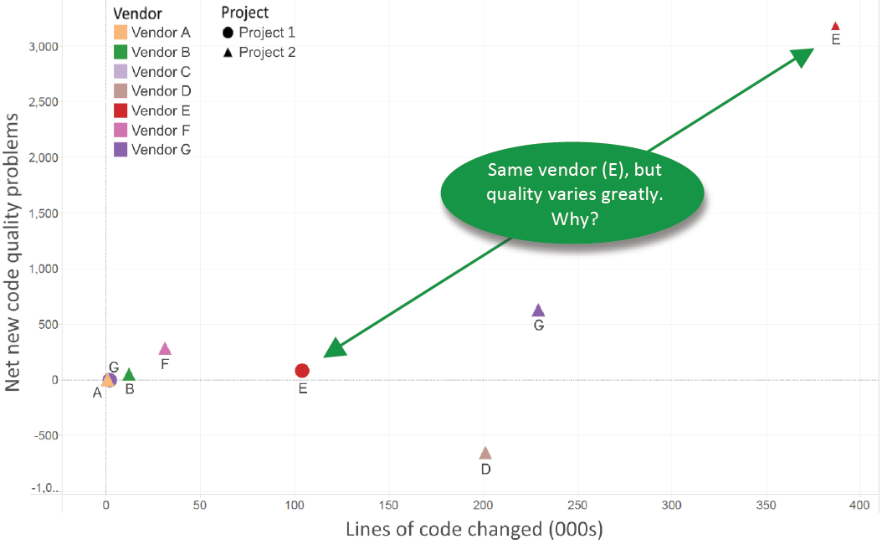
*“Semmle engineering analytics glean insights from the code you create every day to help development teams build software better. Semmle’s engineering analytics platform helps the world’s leading technology companies rapidly identify and respond to critical vulnerabilities and develop secure, high-quality software.” (Semmle.com, 2017)*

Semmle have two major focus points, ‘Analysing coding behavior, not just code’ and ‘Code as data’:

**Analyse coding behavior, not just code**

Built on groundbreaking programming languages and database research, and data science, Semmle enables software engineering teams to gain actionable insights from the code they create. These insights help software leaders make data-driven decisions that improve software delivery, and organizational development and efficiency.

**Code as data**

****The code change history in your software repositories can speak volumes about how well your teams develop software. Semmle provides the unique ability to convert your code into a knowledge base that can be explored and provide you with feedback on how well your organization works.

**How Semmle works**

1. Semmle converts the source code change history in your software repositories into a knowledge base. Source code is deconstructed into a data model that interrelates the specific element types that compose your code, such as methods, expressions, variables, and so forth, with their location in the source code. This makes it possible for you to interrogate your code base in any way you can imagine to understand your code and how it was created.
2. You can further enrich the knowledge base with other contextual data. This can include additional developer detail like job title, location, employer, cost, seniority, and skills. It can also include project-level information like bug reports. The more context you add, the more insight you can gain.
3. Dashboards display visuals about your project portfolio, such as coding activity and quality over time. You can drill down into specific projects to view activity history for individual project team members.
4. Ad-hoc code exploration and repo mining are possible using Semmle QL, a declarative, object-oriented query language based on Datalog. QL queries can search your code base narrowly or widely for any syntax, or logic or data flow, no matter how complex. QL is an invaluable tool for architects and developers, quality assurance, and security teams.

## **CodeClimate:**

This computational platform is used by over 100,000 projects. They analyse on average 2 billion lines of code daily. Code Climate incorporate fully- configurable test coverage and maintainability data throughout the development workflow, making quality improvement explicit, continuous and ubiquitous. Some features provided by this platform are:

* Automated Git Updates- Nothing to install. Code Climate runs every time a new commit is pushed.
* Activity Feeds- Up-to-the-minute information so a company can see when and how code changes.
* Instant Notifications- Major security and quality changes pushed to where employees work: email, Campfire, HipChat, and RSS feeds.
* Team Sharing- Instant access for a whole team to maximize code visibility across projects.
* Duplication Detection- Fuzzy matching algorithm finds DRY-violations that human reviewers might miss.
* Email Notification- Instant email notifications to let a company can know when new security and code issues arise
* Security Dashboard

## **Code Coverage Platforms:**

Code coverage is a method of analysing a software engineer’s output and there are many open source code coverage tools online such as CodeCover. Code coverage is an important measurement as we also famously hear of IT failures in the news, such as:

(2004) - UK Destroys Tax Records, costing at least £85m.

(2004) - Ford and Oracle scrap Purchasing System, costing $400m

(2007) - FBI Virtual Case Files Scrapped, costing $170m

(1962) - Rocket Failure for Missing Hyphen, costing $135m in today's dollars.

These are just a few cases. There are likely dozens or hundreds of errors on this scale every year, and likely hundreds to thousands of projects in the <= $1m range. A lot of this is due to a lack of good testing. (Barnes and Barnes, 2017)

The main issue with using code coverage as a form of measurement is 100% coverage does not mean that it has been well tested. It is however fair to say 0% means that it has not been tested at all. In a strict sense, it is not fair to make any claims until the quality of the test suite is established. Passing 100% of the tests isn't meaningful if most of the tests are trivial or repetitive with each other. The question is: In the history of the project, did any of those tests uncover bugs? The goal of a test is to find bugs. And if they didn't, they failed as tests. Instead of improving code quality, they might only be giving you a false sense of security.

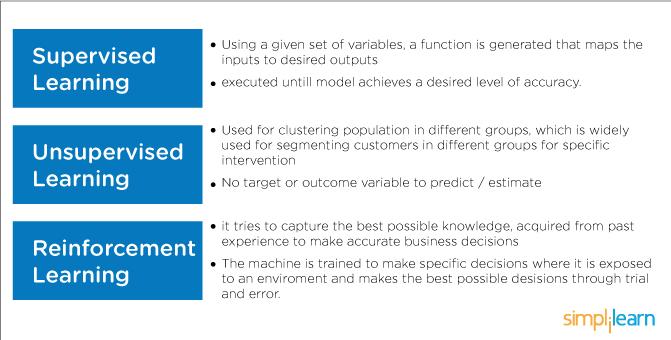
CodeCover is an open source glass-box testing tool for Java and COBOL. Glass box testing is a “testing technique that examines the program structure and derives test data from the program logic/code.” CodeCover measures statement, branch, loop, and strict condition coverage. It is well integrated with a host of development and testing tools including Ant, Jenkins, JUnit, Eclipse and it is licensed under the Eclipse Public Licence (EPL).

More Code Coverage platforms can be found at: https://stackify.com/code-coverage-tools/

# What Algorithm?

I will discuss the process of measuring software engineering paying heed to the relevant algorithms that are available which can analyze the data we have collected. I will mainly discuss machine learning processes and how it aims to provide techniques that improve the state of data. It is no doubt that the sub-field of machine learning has increasingly gained more popularity in the past couple of years. As Big Data is the hottest trend in the tech industry at the moment, machine learning is incredibly powerful to make predictions or calculated suggestions based on large amounts of data. Some of the most common examples of machine learning are Netflix’s algorithms to make movie suggestions based on movies you have watched in the past or Amazon’s algorithms that recommend books based on books you have bought before. Machine learning has the advantage of being unbiased, whereas experts instinctively use their intuition and expertise, which may be biased.

Machine learning algorithms can be divided into three broad categories: supervised learning, unsupervised learning and reinforcement learning. I will illustrate below how easy it is to use these algorithms when measuring a software engineer.



**Supervised Learning:**

Supervised learning regroups different techniques which all share the same principles:

* The training dataset contains inputs data (your predictors) and the value you want to predict (which can be numeric or not).
* The model will use the training data to learn a link between the input and the outputs. Underlying idea is that the training data can be generalized and that the model can be used on new data with some accuracy. (learning, 2017)

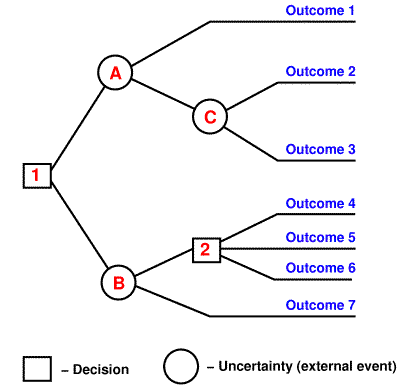
Simply put, supervised learning is useful in cases where a property (label) is available for a certain dataset (training set), but is missing and needs to be predicted for other instances.

Some supervised learning algorithms:

* Linear and logistic regression
* Support vector machine
* Naive Bayes
* Neural network
* Gradient boosting
* Classification trees and random forest

Supervised learning is often used for expert systems in image recognition, speech recognition, forecasting, and in some specific business domain (Targeting, Financial analysis etc)

**Decision Tree Learning:**

A decision tree is a decision support tool that uses a tree-like graph or model of decisions and their possible consequences, including chance-event outcomes, resource costs, and utility. The image to the right illustrates how decision trees are modelled.

From a business decision point of view, a decision tree is the minimum number of yes/no questions that one has to ask, to assess the probability of making a correct decision, most of the time. As a method, it allows you to approach the problem in a structured and systematic way to arrive at a logical conclusion.

Decision tree learning uses a decision tree (as a predictive model) to go from observations about an item (represented in the branches) to conclusions about the item's target value (represented in the leaves). It is a predicitive modelling approach. Tree models where the target variable can take a discrete set of values are called classification trees; in these tree structures, leaves represent class labels and branches represent conjunctions of features that lead to those class labels. Decision trees where the target variable can take continuous values (typically real numbers) are called regression trees.

In decision analysis, a decision tree can be used to visually and explicitly represent decisions and decision making. In data mining, a decision tree describes data (but the resulting classification tree can be an input for decision making). A decision trees can be created to illustrate the process that a software engineer has taken or will take in the project that they are currently undertaking. It can then be seen whether or not the engineer is on track for completing the project in the agreed time. The model can incorporate decisions that may need to be made down the line and illusrate to the customer when these decisions need to be made and the consequences of these decisions. Decision trees are great for agile developers as they are easily ammended and visualised as the project changes and expands. When the project has been completed they are an excellent timeline for the job and one can then analyse how the developers worked, how long certain aspects of the project took them and their final output. Trees combined with customer feedback may be a way to analyse the quality and quantity of developers’ output.

**Unsupervised Learning:**

Dissimilar to supervised learning, unsupervised learning does not use output data (at least output data that are different from the input). Unsupervised algorithms can be split into different categories:

* Clustering algorithm, such as K-means, hierarchical clustering or mixture models. These algorithms try to discriminate and separate the observations in different groups.
* Dimensionality reduction algorithms (which are mostly unsupervised) such as PCA, ICA or autoencoder. These algorithms find the best representation of the data with fewer dimensions.
* Anomaly detections to find outliers in the data, i.e. observations which do not follow the data set patterns.

Most of the time unsupervised learning algorithms are used to pre-process the data, during the exploratory analysis or to pre-train supervised learning algorithms. It is useful in cases where the challenge is to discover implicit relationships in a given unlabeled dataset (items are not pre-assigned).

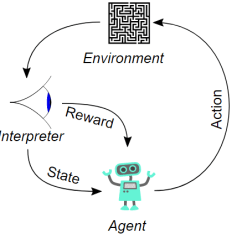
**Reinforcement Learning:**

Reinforcement learning algorithms try to find the best ways to earn the greatest reward. Rewards can be winning a game, earning more money or beating other opponents. They present state-of-art results on very human task, for instance, a paper from the University of Toronto shows how a computer can beat human in old-school Atari video game. (Cs.toronto.edu, 2017)

Reinforcement learnings algorithms follow the different circular steps:

Given its and the environment’s states, the agent will choose the action which will maximize its reward or will explore a new possibility. These actions will change the environment’s and the agent states. They will also be interpreted to give a reward to the agent. By performing this loop many times, the agents will improve its behavior.

Reinforcement learning already performs wells on ‘small’ dynamic system and is definitely to follow for the years to come.

Reinforcement learning falls between the two other learning extremes — there is some form of feedback available for each predictive step or action, but no precise label or error message.

# Ethics

Many believe the conventional wisdom is that Engineering teams can not be measured effectively so we shouldn’t even try. Martin Fowler, an international speaker on software development, thinks it’s a fools errand:

*“I can see why measuring productivity is so seductive. If we could do it 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 to our ignorance.”*

Joel Spolsky, the author of the software development blog ‘Joel on Software’ agrees:

*“Let’s start with plain old productivity. It’s rather hard to measure programmer productivity; almost any metric you can come up with (lines of debugged code, function points, number of command-line arguments) is trivial to game“*

While I hold these two legends of our industry in high esteem, I respectfully disagree. Software engineering teams can be measured, it’s just a little complicated, and because of that we must examine the ethics surrounding the analysis of these eningeers.

Managers often use staffing measurements to conduct performance appraisals, which determine the level of employee performance. Managers also use performance appraisals to reward positive employee performance and suggest improvement in areas where employees are lacking. As previously discussed, there are several measurement options are available for determining engineers’ performance levels. Business owners and managers must act ethically when measuring staff performance because many elements of work life hinge on the results, such as pay increases, promotions, demotions, layoffs and firings.

Ethics is defined as ‘a set of moral issues or aspects (such as rightness)’ (Merriam-webster.com, 2017). Ethics is a very important element that management must consider when analysing a engineer’s performance since anctually measuring an engineer’s output is so arduous.

In "Human Resource Management," Byars describes ways to ensure legal and ethical compliance when conducting staff measurements for performance appraisals. Business owners and managers can take steps to ensure ethical and legal compliance in staff measurement. Reviewing managers should base the measurements in data, time spans and documented events. Managers should also be trained in accepting diversity, and trained in accepted appraisal procedures that apply legal and ethical staffing measurements to performance appraisals.

**Story Points**

I previously discussed SCRUM, which was a form of agile software development. Story points are a unit of measure for expressing an estimate of the overall effort that will be required to fully implement a product backlog item or any other piece of work. (Cohn, 2017) Simply put they are a measure of effort and risk. If we have consistent story points, and figure out how many story points each developer finishes per sprint, then we can extrapolate developer performance. Let's see what happens:

If they finished less than they did last sprint, they're chastised. They are again reminded that they themselves committed, no matter what. Even if you had to help a prod issue, or were in a car accident, or got sick -- you committed. So developers start sandbagging to avoid this. Sandbagging is when an engineer chooses (on purpose) to not play their best i.e. They do not work to full capacity in the fear that during some sprint, something will happen so that they are unable to work to their full capacity and are penalised because of it.

If they finished exactly right, the managers will think the developers finished early and were sitting idle, or were padding their estimates. This leads to frustration and resentment. Alternatively, a perfect finish might be seen as a state where, if everybody worked a few more hours, we'd see more output.

If they finish with more points than they took on, managers will accuse the developers of sandbagging. Then they told that they must accept more points next sprint, to take this into account. That, or you have a "level-setting meeting" where everybody re-agrees what the points represent. This leads to frustration and resentment, not to mention the drop in productivity related to figuring out the new point system.

If a manager asks for doubled productivity, that's easy: double the story-point estimate.

Story points also aren't consistent between developers. Even if everybody agrees that it's a 3-point story, based purely on effort and risk, the wall-time delivery will be different depending on who picks it up. One developer who is intimately familiar with that code may be able to finish in 2-3 hours, while a new junior developer may struggle for 1-2 days. This is proof that we've decoupled productivity from points, and why it's a bad, unfair and unethical metric.

On the official Scrum forums, practioners always have to explain why story points are not a measure of productivity. The Scrum Alliance even has a whitepaper called The Deadly Disease of Focus Factors, and here is the opening statement of the document:

To check your organizational health, answer these two questions:

1) Do you estimate work in “ideal” hours?

2) Do you follow up on your estimates, comparing it to how many “real” hours work it actually took to get something done?

If so, you may be in big trouble. You are exhibiting symptoms of the lethal disease of the “focus factor”. This is how the illness progresses:

Speed of development will keep dropping together with quality. Predictability will suffer. Unexpected last moment problems and delays in projects are common. Morale will deteriorate. People will do as they are told, but little more. The best people will quit. If anything gets released it is meager, boring and not meeting customer expectations. As changes in the business environment accelerate, the organization will be having trouble keeping up. Competitors will take away the market and eventually the end is unavoidable.

This is an example of why examining the ethics surrounding the measurement you chose is so important. If you choose story points, you run the risk of demoralizing your employees and creating an air of resentment towards management.

**Productivity:**

This is probably the most controversial element of a software engineer to measure. Many people in the agile community will argue that productivity shouldn’t be measured as it’s counter to the agile philosophy. You should instead track velocity to get an idea of when things will be delivered. Say for example a firm practices SCRUM, and has two week sprints iterations, and based on historic velocity they have a decent idea of what they can get done in an iteration. What is often clear in SCRUM teams is that it takes super-human focus to get things across the finish line. And that is usually not conveyed in user stories. The human focus here is instrumental in sucessfully completing the sprint and what can truly mearure human focus and input?

# Conculsion

When we measure anything about a person we must always remeber that they are a person. Not a name on a page. Not a data point on a graph of thousand of points. Each individual is different, has different lives, backgrounds and experience. A fresh faced graduate will not have the same experience as an engineer who has been in the industry for twenty years, and simillarly an engineer who has been in the industry for twenty odd years will not have the up to date knowledge on new, emerging technologies that a graduate will. It is by no means fair or ethical to “measure” these people to the same stadards. Is there an algorithm or computational platform that can take into account someone having a bad day when measuring their productivity? Computers are great but it is impossible for them to understand and consider human elements such as creativity and imagination. How can these two vitally important aspects of software engineering be measured?

At the end of the day, the data that can be gathered and analysed as previously discussed above is very useful but whenever it is being used, the user must also consider the human aspects of the engineer the data applies to and remember every piece of data relates to a person, similar to him or her. A manager generally knows who his/her best developer is. When analysing developers, trust your gut. Even though you can't just put numbers on it, most developers find it easy to spot good and bad developers. There's just something telling you that they're better. It could be the way they talk about their technology, the thought they put into a solution’s description, the comments in the code, or the code itself. The output of an engineer is subjective.

"You can't plan if you can't measure." This is an idea still taught in business school, it's a mantra of many managers, and it's wrong in this context. It assumes everything a developer does is objectively and consistently measurable. As we've shown above, there still doesn't exist a reliable, objective metric of developer productivity. I posit that this problem is unsolved, and will likely remain unsolved. The smartest minds of Microsoft, Amazon, IBM, Intel, Wall Street, the Bay Area, Seattle, New York, and London still haven't found that magical metric.

Many dont truly understand what a software engineer does. When a client asks, "can you tell me how many people visited our site yesterday and clicked on the newsletter signup?", it sounds like a simple request. You just take all the people, find the ones who clicked the thing, and count it. But, let's take a dev perspective. How do we identify visitors? Is IP good enough? Do we support IPv6? Do we want to use cookies? Is our cookie policy legally compliant in Europe? Do we have to worry about COPPA? Do we want to de-dupe visitors? How do we track that people clicked on a link? What's the implication of click-stream tracking? Will our infrastructure support that? How important is accuracy? If we lose one click record, does that matter? This is what developers do. For every line of code written, they we are answering all of these questions in excruciating detail. Computers have no context and when measuring an engineers productivity, it cannot take into account all the possibilities the engineer has to take into account before even writing a line of code, as illustrated above.

In conclusion, I believe that the measurement of software engineering as a process is one that brings with it a huge amount of questions. Questions such as what data is the best to measure? How should one collect such data? Where should one store this data? How one can study this data? And, finally, is the collection and examination of this data ethical? One should approach this area with caution. Such questions blur the lines between ethical and unethical, between right and wrong. The field of software engineering is one that has undergone rapid expansion in the past decade. ‘Artificial Intelligence’ and ‘Machine Learning’ are buzz words that are becoming increasingly common today. However, it appears that few genuinely understand the true meaning of them and the potential they have to alter our world irreversibly. And, while the principal component of such change is for the good, there is inevitably an element of harm. It is alarming, the extent to which humans can be tracked and analyzed in the modern world today. At the end of the day, the responsible implementation of this power is up to us and it is our responsibility to dicsuss and examine all of the possible implications of these.

# Bibliography:

Anon, (2017). [online] Available at: https://www.quora.com/How-should-a-software-engineers-productivity-be-measured [Accessed 14 Dec. 2017].

Emptyeasel.com. (2017). Rebecca Elfast: Expressive Watercolor Paintings with Few Brushstrokes. [online] Available at: http://emptyeasel.com/2009/05/06/rebecca-elfast-expressive-watercolor-paintings-with-few-brushstrokes/ [Accessed 14 Dec. 2017].

Cs.stanford.edu. (2017). Crunch Mode: programming to the extreme - The Relationship Between Hours Worked and Productivity. [online] Available at: http://cs.stanford.edu/people/eroberts/cs201/projects/crunchmode/econ-hours-productivity.html [Accessed 14 Dec. 2017].

week, F. (2017). Ford factory workers get 40-hour week - May 01, 1926 - HISTORY.com. [online] HISTORY.com. Available at: http://www.history.com/this-day-in-history/ford-factory-workers-get-40-hour-week [Accessed 14 Dec. 2017].

Inc.com. (2017). Stop Working More Than 40 Hours a Week. [online] Available at: https://www.inc.com/geoffrey-james/stop-working-more-than-40-hours-a-week.html [Accessed 14 Dec. 2017].

Wiki.c2.com. (2017). Lines of Code. [online] Available at: http://wiki.c2.com/?LinesOfCode [Accessed 14 Dec. 2017].

Scrumguides.org. (2017). Home | Scrum Guides. [online] Available at: https://www.scrumguides.org/ [Accessed 14 Dec. 2017].

Code Red. (2017). You Can’t Measure Software Engineering Productivity, so Measure Job Satisfaction Instead. [online] Available at: https://redfin.engineering/measure-job-satisfaction-instead-of-software-engineering-productivity-418779ce3451 [Accessed 14 Dec. 2017].

Csdl.ics.hawaii.edu. (2017). LEAP | Collaborative Software Development Laboratory. [online] Available at: http://csdl.ics.hawaii.edu/research/leap/ [Accessed 14 Dec. 2017].

Csdl.ics.hawaii.edu. (2017). Hackystat | Collaborative Software Development Laboratory. [online] Available at: http://csdl.ics.hawaii.edu/research/hackystat/ [Accessed 14 Dec. 2017].

Semmle.com. (2017). Semmle | Software engineering analytics. [online] Available at: https://semmle.com/ [Accessed 14 Dec. 2017].

Barnes, D. and Barnes, D. (2017). The Myth of Developer Productivity. [online] Dev9. Available at: https://dev9.com/blog-posts/2015/1/the-myth-of-developer-productivity [Accessed 14 Dec. 2017].

Cs.toronto.edu. (2017). Playing Atari with Deep Reinforcement Learning [online] Available at: https://www.cs.toronto.edu/~vmnih/docs/dqn.pdf [Accessed 14 Dec. 2017].

learning, M. (2017). Machine Learning Explained: supervised learning, unsupervised learning, and reinforcement learning - Enhance Data Science. [online] Enhance Data Science. Available at: http://enhancedatascience.com/2017/07/19/machine-learning-explained-supervised-learning-unsupervised-learning-and-reinforcement-learning/ [Accessed 14 Dec. 2017].

Merriam-webster.com. (2017). Definition of ETHIC. [online] Available at: https://www.merriam-webster.com/dictionary/ethic [Accessed 14 Dec. 2017].

Cohn, M. (2017). What Are Story Points?. [online] Mountain Goat Software. Available at: https://www.mountaingoatsoftware.com/blog/what-are-story-points [Accessed 14 Dec. 2017].