Measuring Software Engineering

Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

Cs3012 – Software Engineering

Seán Fitzpatrick | fitzpas5@tcd.ie | 16318530

Software Engineering

Software engineering is a complex profession and as such it can be difficult to measure its production. Unlike many other engineering disciplines, its output is not physical but instead is the conception and implementation of intangible software systems. While the tasks carried out by software engineers vary by sector, position and role, overall the main components of their work involve **Requirement Definition**, **System and Software Design**, **Implementation**, **Testing** and **Maintenance** of software systems (Shown in Fig A). Each of these tasks are very different, but all play a crucial role in the software development process.

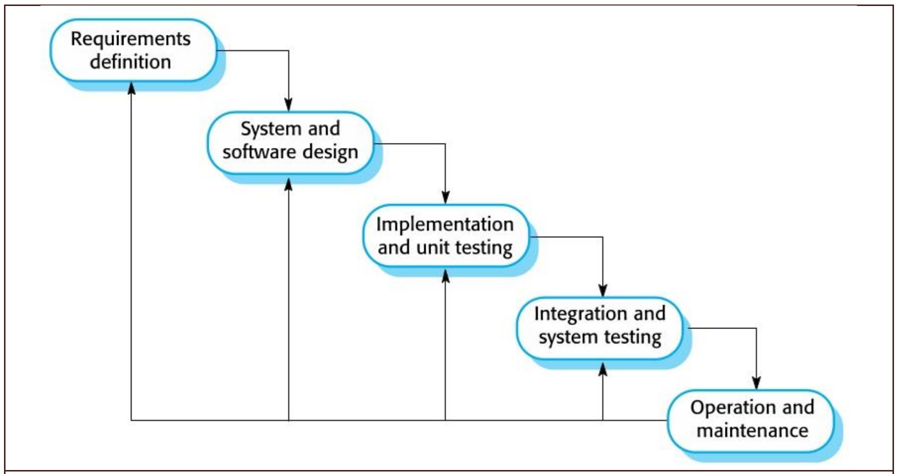

Requirements definition
System and software design
Implementation and unit testing
Integration and system testing
Operation and maintenance

Fig A: Stages of the Waterfall Software Development process

Measuring a Software Engineer's performance requires the gathering and analysis of data concerning these tasks. This analysis can be beneficial for not just management but also for the individual software engineers themselves. Management can use these measurements for important decisions like planning time schedules for development tasks, determining recommended practices and coding standards, selecting technologies and toolsets, etc. Individual Software Engineers can use the analytics to improve their skills and productivity. This information can identify a developer's strengths and weaknesses to recommend the areas that they can improve.

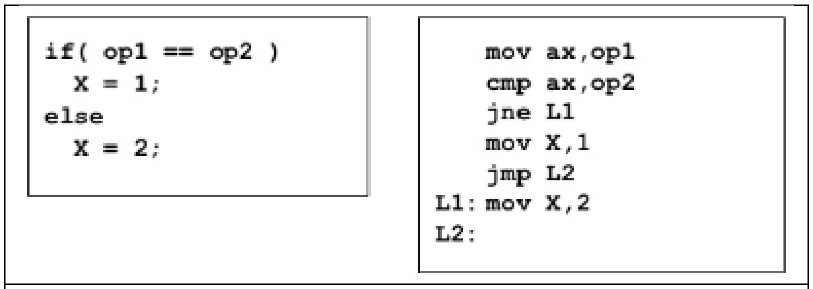
The requirement for developers to preform optimally is a goal that is of great concern by the majority of businesses operating in this industry. One of the reasons for this priority is due to the global shortages of software engineers. Many companies are struggling to hire more developers. As such company's productivity is hampered by the limited number of software engineers that they can hire. Due to this many company's aiming to improve productivity instead measure their employees work and identify the ways that they can be as efficient as possible.

"Our policy is literally to hire as many talented engineers as we can find. The whole limit in the system is just that there aren't enough people who are trained and have these skills today." -Mark Zuckerberg (Dickey, 2013)

The process of measuring software engineer's performance is very important. I will examine this process in detail and in particular I will discuss the data used for this analysis, the platforms available for this analysis, the algorithmic approaches taken and the ethical concerns of this analysis.

Measurable Data

In software engineering, metrics are the standards and systems of measurement of a software engineers performance. These metrics are very important as they can describe, summarize and evaluate the software product of an engineer's work process. There are two generally types of software engineering metrics, **Product metrics** and **Process Metrics**. Product metrics are used to describe the quality of the code. It is used to determine the size of the products, its complexity, how reusable it is and how easy it is to maintain. Process metrics looks at the engineers work process. It is important as it is used to identify successful work processes. Businesses can look at how the best engineer's carryout their work and encourage these processes or if they notice that certain processes are reducing productivity they can discourage them. (Sillitti, Janes, Succi, & Vernazza, 2003)



```
if( op1 == op2 )
  X = 1;
else
  X = 2;
```

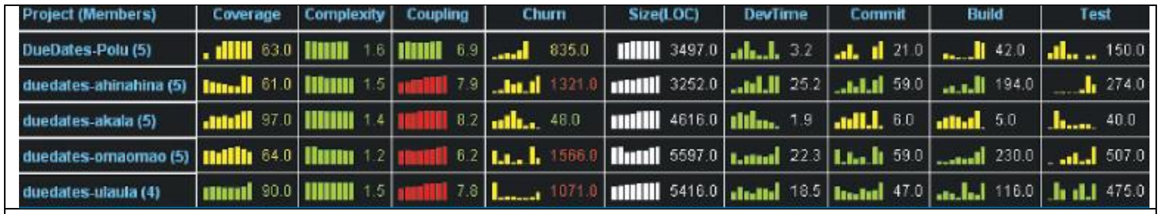
```
    mov ax,op1
    cmp ax,op2
    jne L1
    mov X,1
    jmp L2
L1: mov X,2
L2:
```

Fig B: Lines of code in Java and Assembly for an if-else statement. (2008, سعد)

The history of software engineering metrics is almost as old as software engineering itself. In the 1960's various product metrics were developed to measure and evaluate code characteristics. These were crude methods that reduced complex code down to easily understandable and comparable figures. These measures were fundamentally based on the Lines Of Code(LOC) or thousand Lines Of Code(KLOC) in a program. A software engineers productivity would be measured as LOC or KLOC per month, the quality of their code would be assessed by the defects per KLOC. In 1976 Thomas McCabe formulated a measure to evaluate a programs complexity (Cyclomatic Complexity). This measure is quantifying a programs complexity into a figure based on the amount of linear independent paths that it has. These metrics can be very useful and are widely used in industry today, however they have several flaws. In the 1970s there was a clear need for further metrics as there was a new diversity of programming languages. Many of these languages were high level and could perform the same functionality with less lines of code. This became an issue as a Software Engineer's output would be devalued if they used a higher-level language like Java rather than a lower level language like assembly code as there would be fewer lines of code needed (E.g. see Fig B). Further metrics on a Software engineers process were needed in addition to these metrics. They are still useful today but should not be used in isolation. (Fenton & Neil, 1999)

Software process metrics were developed in order to get a more detailed view into a software engineers work process. Early primitive metrics involved looking at simple metrics like hours spent coding, documenting, testing etc. These metrics were often recorded and analyzed manually. This created large overhead costs in time for both management and developers. The cost of this gathering and analysis process also limited the type of metrics that could be recorded. In addition to this, many developers often avoided recording these simple metrics as they felt it was not an important activity. The development in automated tools for collecting and analyzing metrics have facilitated developers to more easily collect these simple process metrics as well as more granular and detail process metrics that were not possible before automating this process. Information could now be collected every minute or even every second. Tools like PROM or Hackystat can be incorporated into an engineer's toolsets and record process metrics in the background while they work. New metrics where now feasible to collect like editing time, number and type of changes in a class or in a file, how often they add a test case or change a class and even the number of times the engineer changes their active buffer. This automated process also reduces the human errors related manually collecting the data. These metrics have the ability to record the work processes of the most productive developers and identify processes the create inefficiencies so as to recommend best practices to other engineers. This also facilitates metrics to be collected on groups of engineers or projects as a whole to view how effective engineers work together. (Sillitti, Janes, Succi, & Vernazza, 2003)

With a combination of both product and process metrics engineers and management can accurately measure and evaluate software products and processes with and effortless and understandable procedure. (E.g. Fig C shows some of the detailed metrics accessible from Hackystat).



Project (Members)	Coverage	Complexity	Coupling	Churn	Size(LOC)	DevTime	Commit	Build	Test
DueDates-Polu (5)	63.0	1.6	6.9	835.0	3497.0	3.2	21.0	42.0	150.0
duedates-ahinahina (5)	61.0	1.5	7.9	1321.0	3252.0	25.2	59.0	194.0	274.0
duedates-akala (5)	97.0	1.4	8.2	48.0	4616.0	1.9	6.0	5.0	40.0
duedates-omaomao (5)	64.0	1.2	6.2	1566.0	5597.0	22.3	59.0	230.0	507.0
duedates-ulaula (4)	90.0	1.5	7.8	1071.0	5416.0	18.5	47.0	116.0	475.0

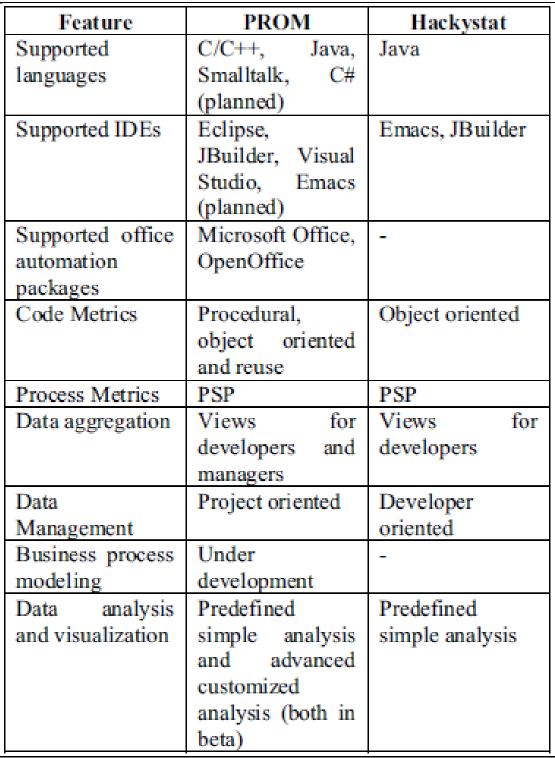
Fig C: Detailed metrics available from Hackystat (Johnson, 2013)

Computational Platforms Available

There have been various tools, processes and methodologies developed for the measurement of a software engineers performance. These platforms have been widely adopted in industry by software engineers and management for the common goal of maximizing a developer's productivity. One of the first methodologies for this process was the **Personal Software Process (PSP)**. This allowed engineers to measure their work by manually recording and analyzing various simple metrics in a spreadsheet format. This process was extremely flexible as developers were easily able to add additional metrics on any factor that they believed was relevant (E.g. a developer can include the number of interruptions they had per day). The PSP's creator, Watts Humphrey strongly encouraged engineers to modify the forms and procedures to fit their personal circumstances. However, the PSP had several critical flaws. This process was very time consuming with some developers having to manually fill in 12 forms. This recording reduced the time that software engineers could spend on development processes. In addition, the information gathered often included recording and calculation errors due to the manual aspect of the analysis.

One organization that aimed to create a system to reduce the overhead of the PSP was the Collaborative Software Development Laboratory (**CSDL**) at the University of Hawaii. They developed the **LEAP** toolkit. This software aims to circumvent the calculation errors associated with the PSP by automating and normalising this process. While data still had to be manually entered it reduced the time needed for analysis and provided additional, more complex statistics to the users such as estimations on required development times based on previous records. While this platform greatly reduced the cost of analysis it also made the PSP less flexible. If a developer wanted to include a new measurement, that was not included in LEAP they would need to develop a new toolset for LEAP rather than just a simple spreadsheet that was previously needed. However, the benefits due to automation outweighed its costs in flexibility and new automated processes of measurements became increasingly popular. (Johnson, 2013)

Further platforms have been developed to measure a Software Engineers performance, many with a stronger emphasis on automating this process so as to reduce the overhead costs involved. One of these platforms is **Hackystat** which was also developed by the **CSDL**. When developing this platform their initial focus was to collect process information on a developer and product information on their output in ways that required little to no time costs for this process. Hackystat would attach sensors to the engineer's work tools and automatically collect data in the background while they worked requiring no effort for this collection. These sensors could be integrated into the developer's editor tools, testing tools build tools, server tools etc. The platform would retrieve the information from the different work tools and ensemble them together to provide a wholistic view of the engineers work process. With this platform the developer would no longer have to waste time manually collecting and analysing this data and could fully focus on their development tasks. This broke the loops that developers faced where they would do some work and the have to stop in order to record information on that work. With this platform developers could now get an incredibly intimate view of their work processes. Due to its extensive collections new metrics were now feasible to measure that were not feasible if the data was manually collected and inputted. Other platforms have also been developed like **PROM** that pushes the details automatically collected on Software Engineers even further. PROM has other features not present in Hackystat like support for further languages and IDE's, can be incorporated with Microsoft Office tools etc (See Fig D). (Johnson, 2013)



Feature	PROM	Hackystat
Supported languages	C/C++, Java, Smalltalk, C# (planned)	Java
Supported IDEs	Eclipse, JBuilder, Visual Studio, Emacs (planned)	Emacs, JBuilder
Supported office automation packages	Microsoft Office, OpenOffice	-
Code Metrics	Procedural, object oriented and reuse	Object oriented
Process Metrics	PSP	PSP
Data aggregation	Views for developers and managers	Views for developers
Data Management	Project oriented	Developer oriented
Business process modeling	Under development	-
Data analysis and visualization	Predefined simple analysis and advanced customized analysis (both in beta)	Predefined simple analysis

Fig D: PROM & Hackystat feature comparison (Sillitti, Janes, Succi, & Vernazza, 2003)

However, while this automation greatly reduced the costs associated with collection and analysis it also came with drawbacks. As the CSDL observed, there exist a trade-off between collecting detailed and in-depth information with social and political costs of its collection. These tools were so unobtrusive to Software Engineers, that often information was being collected on aspects of their work that they were not aware of. Many engineers were not comfortable with this and the lack of control that they had over what measurements were used to record them. This reduced the adoption of this platform in industry.

There have been several further platforms developed for this measurement, however none have yet found a 'silver bullet' that has managed to break this trade-off rule. In addition, in industry there has been no clear consensus agreed to which approach is optimal, and a business must choose a platform that best suits their needs. I agree with Johnson in his paper, that for a platform to be most effective and adopted, it should have a "hybrid approach that mixes the best of automated collection and analysis with carefully chosen, high-impact manual data entry by developers" (Johnson, 2013). One recent platform that I believe strikes this balance well is **Timeular's ZEI°**. This is an eight-sided die shaped device that allows users to easily measure and analyse their productivity (Fig: E). By simply placing the device on the side associated with your current activity, it will begin tracking the time you spend on the task. In addition to this, the platform provides the user in-depth analytics and reports on how much time they spend on task, to effectively manage their work process (Fig F). I believe that there are three factors that result in this effective balance. Firstly, the device allows for easy automated data collection. The users simply must to put the device facing up on a certain side. Also, the device allows the user to easily manage the data that is collected on them. The device is simplistic enough that the user is constantly aware and can comprehend what metrics are being recorded at any given time based upon which face is currently up. In addition, the user can easily suspend this collection at any given time by placing it on its stand. Lastly the device allows for a great degree of flexibility. It allows you to add your own markers on each side of the device to customise your analysis. The device comes with several generic stickers to record times for activities like calls, coffee breaks, meetings, etc. In addition, it comes with a special marker that allows you to create your own custom activities. For example, you can make a marker on one of the sides to measure the time you spend dealing with office visits per day. This allows for some of the flexibility needed for the PSP. It is because of this balance that I believe this device has become so widely adopted in industry by companies like Facebook and Google.

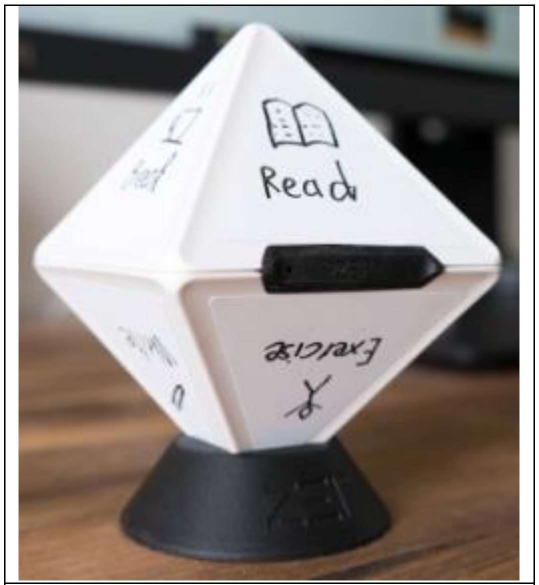
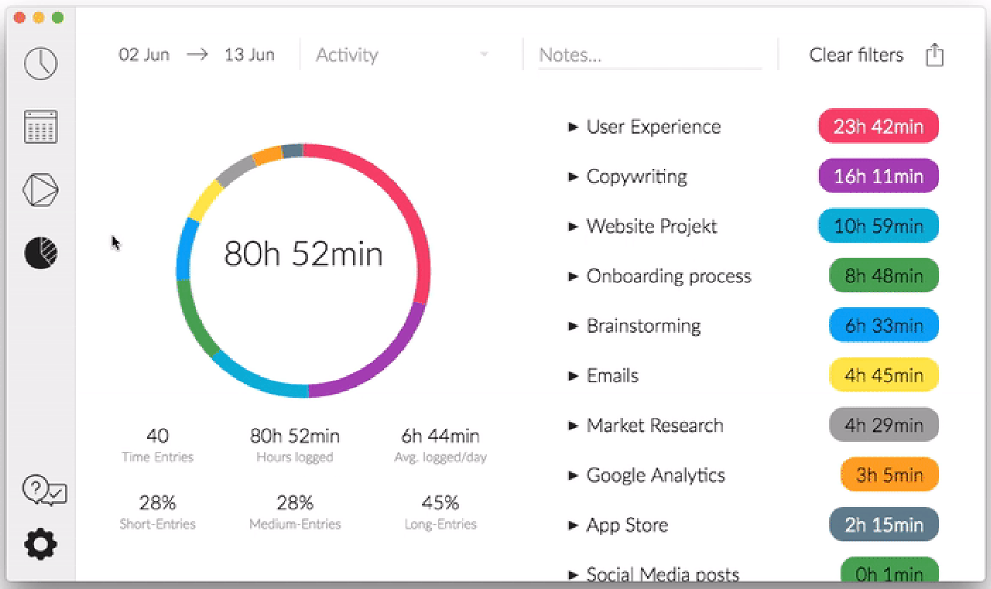


Fig E: Timeular's ZEI°.



02 Jun → 13 Jun | Activity | Notes... | Clear filters

80h 52min

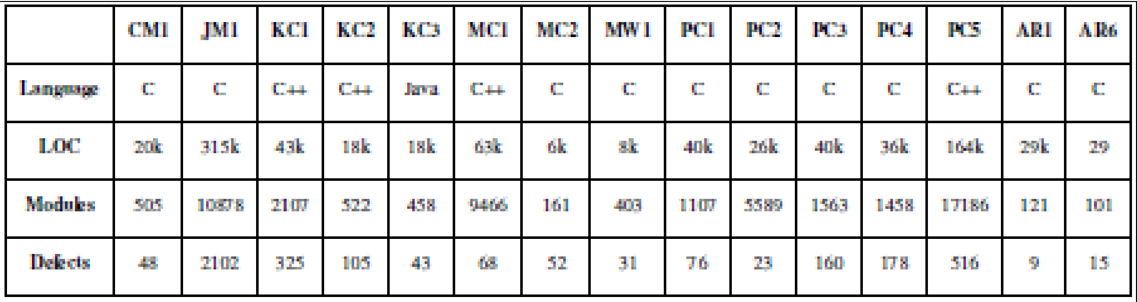
40	Time Entries
80h 52min	Hours logged
6h 44min	Avg. logged/day
28%	Short-Entries
28%	Medium-Entries
45%	Long-Entries

▸ User Experience	23h 42min
▸ Copywriting	16h 11min
▸ Website Projekt	10h 59min
▸ Onboarding process	8h 48min
▸ Brainstorming	6h 33min
▸ Emails	4h 45min
▸ Market Research	4h 29min
▸ Google Analytics	3h 5min
▸ App Store	2h 15min
▸ Social Media posts	0h 1min

Fig E: Timeular's ZEI° time sheet report

Algorithmic Approaches

So far, I have discussed the various types of metrics collected on Software Engineers and the computational platforms used to analyze these metrics. I will now discuss the algorithmic approaches used to measure and analyze this data. An algorithm is a set of rules and procedures used to carry out a certain task. Here this task is to evaluate and measure a software engineers processes and output. Due to the complexity of a software engineers work it is very difficult to boil this process down into one simple formula. To create a model for this analysis that is sufficiently accurate, various Machine Learning Algorithms have been adopted to analyze both product and process metrics. Machine Learning algorithms involve creating a predictive model with the use of real world data. These algorithms are capable of



	CM1	JM1	KC1	KC2	KC3	MC1	MC2	MW1	PC1	PC2	PC3	PC4	PC5	AR1	AR6
Language	C	C	C++	C++	Java	C++	C	C	C	C	C	C	C++	C	C
LOC	20k	315k	43k	18k	18k	63k	6k	8k	40k	26k	40k	36k	164k	29k	29
Modules	505	10878	2107	522	458	9466	161	403	1107	5589	1563	1458	17186	121	101
Defects	48	2102	325	105	43	68	52	31	76	23	160	178	516	9	15

Fig F: Dataset details used for classification (Saiqa Aleem, 2015) providing a greater degree of analysis and insight of data then would be possible manually.

Various Machine Learning classifiers have been used to make predictions on whether code contains defects or not, using both supervised and unsupervised learning. This is greatly needed today as with the recent proliferation of software systems and the increasing complexity of them, new automated approaches are needed to accurately detect where defects might be occurring in large code bases. Classifier algorithms involve the use of large descriptive datasets. These sets contain descriptive information on of each of software systems including their LOC, number of modules, programing language, etc. (Fig F). They also contain the class label of each item in the dataset to indicate whether it is defective or not. This dataset is split into two sections, one for training and one for testing. After preprocessing, the training set is fed into the machine learning classifier algorithm and this will create a predictive model that will attempt to classify the group that new entries belong to (I.e. Defective or Non-defective) based on the previous examples it has seen. This is a binary classification. After this the predictive model that was generated is tested on the test set to see how well it preforms on data that it has never seen before. This will give an indication of how well it will perform on real world applications. In the paper Benchmarking Machine Learning Techniques for Software Defect Detection (Saiqa Aleem, 2015) analyzed the performance of various classifier techniques on a dataset of NASA projects and found that the classifiers mostly returned similar accuracy results (Fig G).

One algorithm that I believe is very effective for this measurement is a **Decision Tree**. Firstly, a decision tree is very accurate at correctly predicting whether a software system is defective, receiving an accuracy score of around 88.47% on the training dataset (Saiqa Aleem, 2015). The decision tree algorithm detects whether a given sample is defective by generating a series of rules. These rules are called decision nodes. The rule of each decision node is generated by finding a case where it can separate the two classes as much as possible. This process will continue until the two classes are completely separated or it has reached its max height. The advantage that this algorithm has over others is that you can easily interpret the process the algorithm is taking in order to classify the sample. This is not the case with many other machine learning algorithms which are often seen as a 'black box' where input is given, output is return, and the procedures taken in between is too complex to interpret. The models generated by a decision tree can be easily converted into a tree-based graph so as to understand why the model is making its decision (fig H). I believe that this ability to understand the decision-making process is especially important when measuring the output of Software engineers so as to ensure that the algorithm is not rewarding unwanted behavior and devaluing wanted behavior.

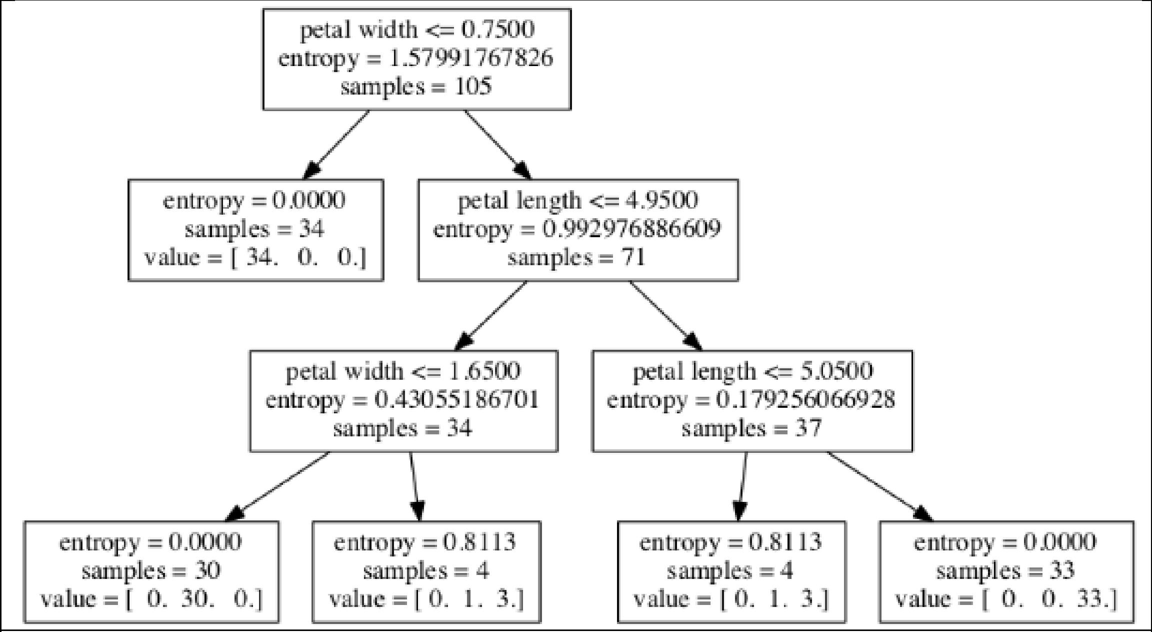


Fig G: Decision tree used to classify the species of a flower using it petal width (Raschka, 2015)

Ethical Concerns

I will now discuss some of the ethical concerns regarding this kind of analysis. Businesses must be careful not to outstep their bounds and drive away their employees. While the measurement of employee's performance is a common practice that can be seen across various industries and is seemingly harmless, businesses should set various restrictions in place in order to reduce the ethical issues that may arise from measuring a software engineer's performance.

Employers must make the fact that employees are being measured abundantly clear to the employees. In addition to this they must also disclose what metrics are being recorded. Employees may become uncomfortable if they are unaware of which metrics are being recorded on them and when. An example of this can be seen with Hackystat. Due to its extensive and detailed collection of data, employees were not able to know what all the metrics being recorded at a given time are. Many engineers were not comfortable using this software because of this, with one user referring to the platform as "hacky-stalk". (Johnson, 2013)

The information gathered by these platforms may contain personal identifiable information. Many developers will not be comfortable with this information being available to anyone in the organisation. Businesses should limit the access of these metrics to only certain individuals that need the information (E.g. Managers would need to have access to this data in order to make important decisions). In addition, employees should have access to all the metrics that are recorded on them. This would lead to a more trusting and transparent workplace as employees will be fully aware of and comprehend the measurement system in place.

Only data that is relevant should be recorded. Businesses should limit the collection to data that is relevant to current goals and not just record every metric that they can get their hands on. Businesses should follow a Goal Question Metric (GQM) approach, which is an approach for selecting metrics on software engineers. Here metrics are chosen based on questions about current business goals (E.g. Does paired programming lead to better software quality). (Victor R. Basili, 1994). This would ensure that businesses are only selecting relevant metrics and employees would know what their data would be used for.

Finally, businesses should avoid solely using quantitative metrics when measuring an employee's productivity. Software engineers have a very complex profession, and as such it is difficult to reduce their entire work down into a few simple figures. Purely focusing on this would leave out crucial factors that can't be as easily evaluated like how well the employee adds to staff morale, do they fit in with the company's culture etc. In addition, by focusing on making employees work in a certain way to maximise chosen productivity metrics, new innovative approaches may be discouraged.

Bibliography

Dickey, M. R. (2013, 2 28). *Mark Zuckerberg, Bill Gates, Jack Dorsey And Other Techies Want More Kids To Learn How To Code*. Retrieved from Business Insider: https://www.businessinsider.com.au/mark-zuckerberg-on-kids-coding-2013-2

Fenton, N. E., & Neil, M. (1999). Software metrics: successes, failures and new directions. *The Journal of Systems and Software*, 149-157.

Johnson, P. M. (2013). *Searching under.* Manoa: IEEE Software.

Raschka, S. (2015). *Python Machine Learning.* Birmingham, United Kingdom: Packt Publishing Limited.

Saiqa Aleem, L. F. (2015, 5). BENCHMARKING MACHINE LEARNING TECHNIQUES. *International Journal of Software Engineering & Applications*, pp. 11 - 23.

Sillitti, A., Janes, A., Succi, G., & Vernazza, T. (2003). Collecting, Integrating and Analyzing Software Metrics and Personal Software. *Proceedings of the 29th EUROMICRO Conference "New Waves in System Architecture" (EUROMICRO'03)* (pp. 1-1). IEEE.

Victor R. Basili, G. C. (1994). *THE GOAL QUESTION METRIC APPROACH.*

)2008. (م, سعد. *Assembly Language Lecture 5*. Slide Share.