**Software Engineering Report** 6thDecemeber 2017

**Zach O’Hagan 14318262**

**Introduction**

The purpose of this report is to examine and assess the ways in which software engineering can be measured in terms of data, the computational platforms available to measure and analyse, the algorithmic approaches available and the ethics concerns surrounding this kind of analytics.

**What is software engineering?**

Before we get into the ways in which we can measure and assess software engineering, firstly I would like to give brief overview of the discipline itself. There are numerous definitions for software engineering but the one I found to be most fitting is, the systematic application of scientific and technological knowledge, methods, and experience to the design, implementation, testing, and documentation of software. The field of software engineering applies the disciplined, structured approach to programming that is used in engineering to software development with the stated goal of improving the quality, time and budget efficiency, along with the assurance of structured testing and engineer certification. Software engineering involves a number of discipline that cover the process of engineering software and certification including: requirements gathering, software design, software construction, software maintenance, software configuration management, software engineering management, software development process management and creation, software engineering models and methods, software quality, software engineering professional practices as well as foundational computing and mathematical and engineering study.

Like any process, in order to identify progress, or in some cases the lack of, one must be able to measure the process and specific elements of that process. The first element that I will examine in terms of measurement and analysis is data.

**Measurable data**

Before I begin to examine the different types of metrics that a software engineering might use to measure performance, first let’s define the difference between a measurement and metric. A measurement “*is an indication of the size, quantity, amount or dimensions of a particular attribute of a product or process”* e.g. the number of errors in a certain system. A metric on the other hand is a “*measurement of the degree that any attribute belongs to a system, product or process”* e.g. the number of errors per person hours. Thus software measurements give rise to software metrics.

Software metrics can be divided into two categories, product and process metrics. Product metrics are used to asses the state of the product, tracking risk and discovering potential problems areas. The focus of process metrics is on improving the long term processes of the team or organisation.

Software metrics are a way of putting a measure on certain aspects of development allowing it to be compared to other work which in turn highlights potential areas for improvement. These values have to be assessed correctly otherwise they will not give accurate measurements. Some examples of metrics include lines of code, test coverage, commit count and bugs.

**Lines of code**

This is a very simple and straight forward method where one simply counts the lines of code that a software engineer produces. This in my opinion is not a very good metric due to the fact that it promotes the idea of spreading code over a number of lines when in actual fact it is thought of as best practice to minimise the number of lines of code without there being a quality trade off.

**Test Coverage**

Test coverages measures the amount of code covered by a set of predefined tests. It is considered good practice to play around with these tests i.e. trying to test against a large rang of outcomes, and have separate software engineers to the ones who wrote the actual code write some tests. Test coverage is extremely important and should be carried out while the code is being developed so any need for changes be identified early and dealt with accordingly. it is usually displayed as a percentage and is a good indicator to how testing has been carried out and if more is necessary.

**Commit Count**

This is some what similar to the lines of code metric, where one simply tracks the amount of times they commit to a certain repository. There are a number of benefits to this kind of metric, one being that smaller regular commits allows for other individuals to track your work more easily than if there was just one large commit. Regular commits also reduce the likelihood of losing large amounts of code.

**Bugs**

This refers to the amount of time a software engineer is spending on both fixing bugs or troubleshooting issues as they arise in a week. It is not referring to the amount of bugs in a software engineers code because there will always be bugs in some shape or form. In my opinion this is a good way to measure performance to a certain extent, but it does have it’s drawbacks which I will discuss later.

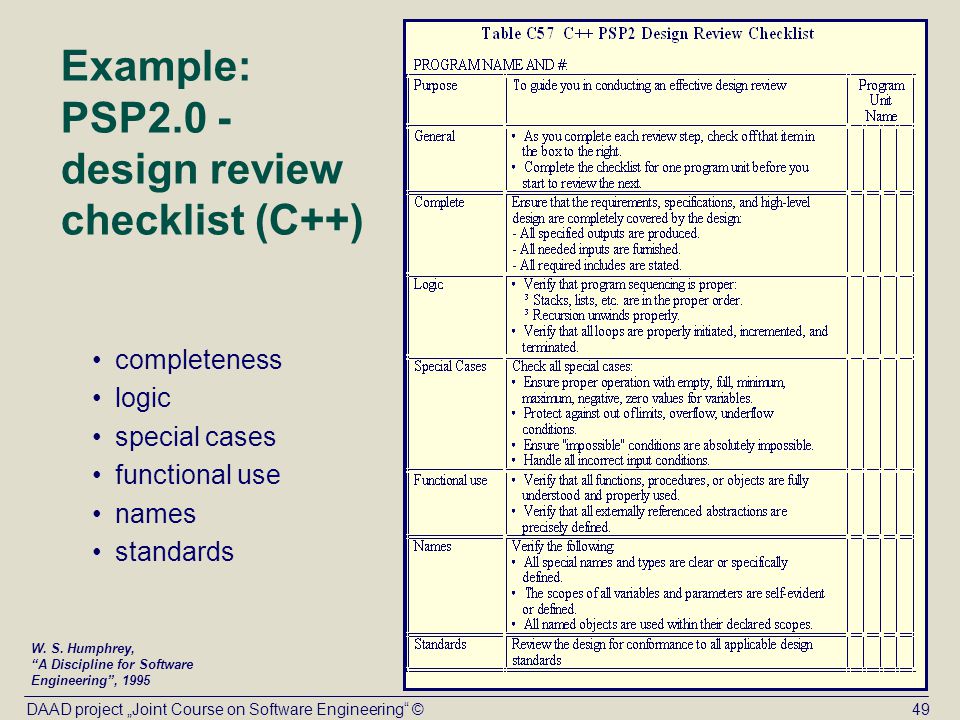
**Computational platforms for measurement and analysis**

**Personal Software Process (PSP)**

One of the first computational platforms used to collect metrics on the software engineering process was the personal software process (PSP). In terms of the measurement of software engineering PSP was quite a significant development. PSP was created by Watts Humphrey to address the need for individual software engineers to acquire a disciplined and effective approach to writing programs. The philosophy behind the PSP is that an organization’s ability to build large-scale software systems dependent upon the ability of its individual software engineers to develop high quality small-scale programs in a disciplined, effective manner. The PSP was created to apply the underlying principles of the Software Engineering Institute’s (SEI) Capability Maturity Model (CMM) to the software development practices of a single developer. The PSP is similar to the Capability Maturity Model (CMM), except that it focuses on the personal process. It was designed to help engineers organize and plan their work, track their performance, manage software defects, and analyse and improve their personal process.

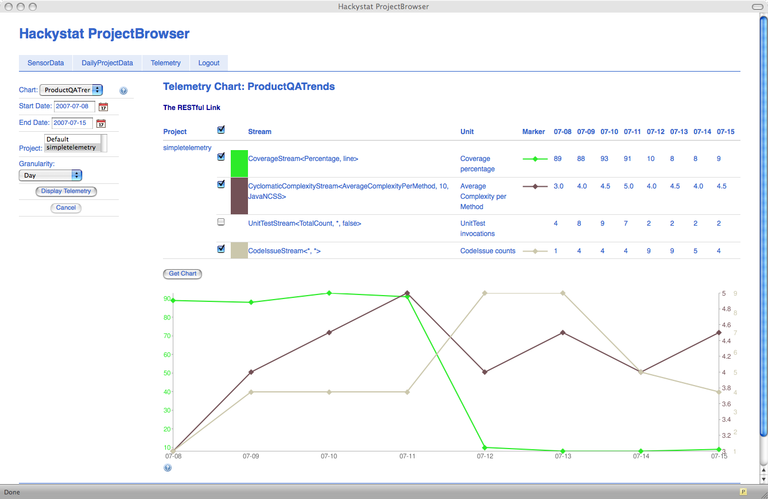
The PSP follows a specific structure of an evolutionary improvement approach. An engineer learning to integrate the PSP into his or her process begins at the first level – PSP0 – and progresses in process maturity to the final level – PSP2.1. Each Level has detailed scripts, checklists and templates to guide the engineer through required steps and helps the engineer improve her own personal software process.

One of the downsides to the PSP is time. All the data that is collected has to be inputted manually into their relevant categories. This requires software developers to commit large and some what unnecessary amounts of time to gathering and logging specific data. When put into practice it may considered inefficient in terms of time.



**Hackystat**

Hackystat is an open source framework for collection, analysis, visualization, interpretation, annotation, and dissemination of software development process and product data. Hackystat users typically attach software ‘sensors’ to their development tools, which unobtrusively collect and send “raw” data about development to a web service called the Hackystat SensorBase for storage. The SensorBase repository can be queried by other web services to form higher level abstractions of this raw data, and/or integrate it with other internet-based communication or coordination mechanisms, and/or generate visualizations of the raw data, abstractions, or annotation. Hackystat does not support certain activities that PSP does such as planning, plan tracking and estimation. It also provides limited data analyse capabilities in comparison to PSP.



**Leap**

Project LEAP and the Leap toolkit are based upon the PSP, and are simply methods developed to automate the PSP. The Leap toolkit collects the same metrics about the development process as the PSP. Both focus on individual developer improvement. Many of the analyses that the Leap toolkit performs come directly from the PSP. To reduce the overhead of collecting and analyzing the data, the Leap toolkit provides several tools for data collection and analysis. Not only does the Leap toolkit reduce the overhead to the user, but by automating the collection and analysis of the data, it eliminates many of the data errors that may be found in the PSP.

**Algorithmic approaches**

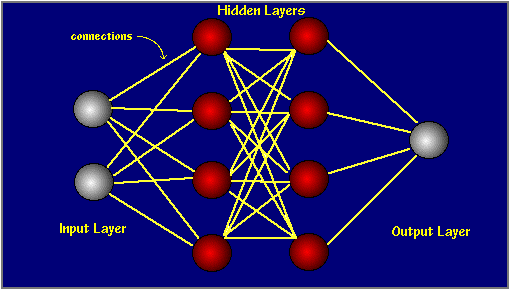
In terms of algorithms that can be used to measure the software process, computational intelligence has a bundle of algorithm that can be used. The expression computational intelligence usually refers to the ability of a computer to learn a specific task from data or experimental observations. Generally, computational intelligence is a set of nature-inspired computational methodologies and approaches to address complex real-world problems to which mathematical or traditional modelling can be useless for a few reasons: the processes might be too complex for mathematical reasoning, it might contain some uncertainties during the process, or the process might simply be stochastic in nature. Many real-life problems cannot be translated into binary language for computers to process it. Computational Intelligence therefore provides solutions for such problems. It uses inexact and incomplete knowledge, and it is able to produce control actions in an adaptive way. CI therefore uses a combination of five main complementary techniques. “*The fuzzy logic which enables the computer to understand natural language,*[*artificial neural networks*](https://en.wikipedia.org/wiki/Artificial_neural_network)*which permits the system to learn experiential data by operating like the biological one,*[*evolutionary computing*](https://en.wikipedia.org/wiki/Evolutionary_computation)*, which is based on the process of natural selection, learning theory, and probabilistic methods which helps dealing with uncertainty imprecision.”*

**Fuzzy Logic**

Fuzzy Logic involves measurements and process modelling made for real life's complex processes. Unlike Artificial Intelligence, which requires exact knowledge, fuzzy knowledge can face incompleteness, and most importantly ignorance of data in a process model. Fuzzy logic is mainly useful for approximate reasoning. The idea of fuzzy logic was first put forward by Dr Lotfi Zadeh in the 1960’s.

**Neural Networks**

Biological neural networks which make up part of the human brain are the inspiration behind neural network computing systems. These networks are made up of learning algorithms , which attempt to identify relationships. Neural networks are typically organized in layers, which are made up of a number of interconnected 'nodes' which contain an 'activation function'. “*Patterns are presented to the network via the 'input layer', which communicates to one or more 'hidden layers' where the actual processing is done via a system of weighted connections*”. The hidden layers then link to an output layer where the answer is output shown in the graphic below.



Facial recognition applications are an example of where neural networks have been used.

**Evolutionary Computations**

Evolutionary computation uses computational models of evolutionary processes as key elements in the design and implementation of computer-based problem solving systems. There are a variety of evolutionary computational models that have been studied. They all share a common conceptual base of “*simulating the evolution of individual structures via processes of selection and reproduction”.* Although simplistic from a biological stand point these algorithms are complex enough to provide a powerful search mechanism.

**Ethics**

There are many definitions of code of ethics and there is no one general one. But it is evident that the all enlighten one crucial thing that It is a formal and systematic statement of rules, principles, regulations or laws, developed by a community of profession to promote its well-being and to exclude or punish any undermining behaviour. Thus all software engineers have the code of ethics as abases over they base their design of systems and their day to day practices on. They are intended to provide guidelines on what should be done and what should not be done while practicing.

There are some ethical concerns that I have identified by using these measurements and metrics. As I outlined earlier in the report I do not think that measuring lines of code is a very good metric, and it could be considered unethical. If a code of ethics is supposed to outline a software engineers duties then the manner in which their progress is measured should be in line with these duties and responsibilities. It seems to me that it would be quite naïve to measure say two developers against each other in terms of the lines of code they write in a given time period. Firstly the standard of the shorter code that is written may be far superior to the other code with more lines. We should also consider the fact that when it comes to software development and writing code, there is not a specified rate at which a developer should be working at i.e. different people will have different approaches to how they write their code and therefore complete the task in different ways and ultimately different speeds. For these reasons I believe from an ethical point of view that measuring lines of code should be avoided.

My second concern would be regarding bugs, which I said earlier are a good measure of performance to a certain extent. The reason for this is similar to my reasons with measuring lines of code, in that bugs arise from mistakes and errors made in an individual developer’s code and as I alluded to in the previous paragraph is most likely different to another developer’s code. So while measuring bugs can be a good measure of performance, the amount of debugging done by a certain developer is dependent on the code that they are have written i.e. one developers code may contain far more bugs than another due to the fact the task they are working on and the method they use are more prone to have bugs arise than their counterparts code and methods, which therefore is not really measuring them on a level playing field.

**Bibliography**

<http://csdl.ics.hawaii.edu/research/hackystat/>

<http://pages.cs.wisc.edu/~bolo/shipyard/neural/local.html>

<http://searchnetworking.techtarget.com/definition/neural-network>

<http://ukci.cs.manchester.ac.uk/intro.html>

<http://whatis.techtarget.com/definition/fuzzy-logic>

<http://whatis.techtarget.com/definition/software-engineering>

<http://www.informit.com/articles/article.aspx?p=30306&seqNum=5>

<http://www.iosrjournals.org/iosr-jce/papers/vol1-issue5/F0152125.pdf?id=2163>

<https://books.google.ie/books?id=MvTmRUm1YSYC&pg=PA80&lpg=PA80&dq=the+aim+of+hackystat&source=bl&ots=Yd4tE_W98C&sig=Ncpbvq6C3T1sa8-hecsB6oUD0rM&hl=en&sa=X&ved=0ahUKEwiWuczh8_DXAhUEzaQKHcq3BH0Q6AEIJzAA#v=onepage&q=the%20aim%20of%20hackystat&f=false>

<https://en.wikipedia.org/wiki/Artificial_neural_network>

<https://en.wikipedia.org/wiki/Computational_intelligence>

<https://en.wikipedia.org/wiki/Evolutionary_computation>

<https://en.wikipedia.org/wiki/Fuzzy_logic>

<https://en.wikipedia.org/wiki/Heuristic_(computer_science)>

<https://en.wikipedia.org/wiki/Software_metric>

<https://link.springer.com/chapter/10.1007/3-540-56602-3_163>

<https://sites.google.com/site/advancedsofteng/software-engineering-as-a-profession/codes-of-ethics-for-software-engineers>

<https://visualstudiomagazine.com/articles/2013/05/01/neural-network-feed-forward.aspx>

<https://www.google.ie/url?sa=t&rct=j&q=&esrc=s&source=web&cd=15&cad=rja&uact=8&ved=0ahUKEwjF4Jyalu7XAhWS-6QKHQYVBz0QFgh0MA4&url=http%3A%2F%2Fresources.sei.cmu.edu%2Flibrary%2Fasset-view.cfm%3Fassetid%3D5283&usg=AOvVaw3IWu7AOYucVV314bL350b8>

<https://www.researchgate.net/profile/Carleton_Moore2/publication/2647145_Investigating_Individual_Software_Development_An_Evaluation_of_the_LEAP_Toolkit/links/5411d1c60cf2bb7347dad82d.pdf>