Aoife Kelly
15318995

The ways in which the software engineering process can be measured and assessed in terms of measurable data

Before beginning this assignment I wanted to try and understand each aspect of the question as much as I could first, this way I could form truthfully my own opinions about each part. I had little to no previous knowledge on this topic and had to do some background research before tackling the essay. By looking at the question we can see that it is clearly divided up into four main parts, measuring the software engineering process, computational platforms available to perform this work, the algorithmic approaches available to us for measuring, and the ethics concerns involved when it comes to analysing software engineers. To understand all parts we need to know that software engineering is the application of engineering to the development, operation, and maintenance of software. We also need to understand the software engineering process in order to fully comprehend the question. The software engineering process is a cycle (see Fig 1). It is split into 5 parts:

Analysis of requirements	To find out the problem and fix the system. To break down the system and analyze each aspect, looking at project goals and figuring out what needs to be created.
Design	To describe in detail the design elements of the new system.
Implementation	Build the system. In this stage it will be important to remember the requirements and the design laid out in the previous stages. Programming code will be written during this stage, as well as unit testing and module testing. This step is intermingled with the next as respective modules will need to be tested before they are integrated with the main project.
Testing/ Verification	Testing is important of course to ensure the the system is functional. People have different opinions as to what the stages of testing are and how much iteration (if any) occurs, but unit, system and user acceptance testings are generally performed.
Maintenance	As lots of things as the world changes the system must change with it.The deployment of a system will always include changes and enhancements before the system is finished. Maintaining the system is very important for example if a key person were to change positions in an organization, new changes would be implemented, which would require system updates.

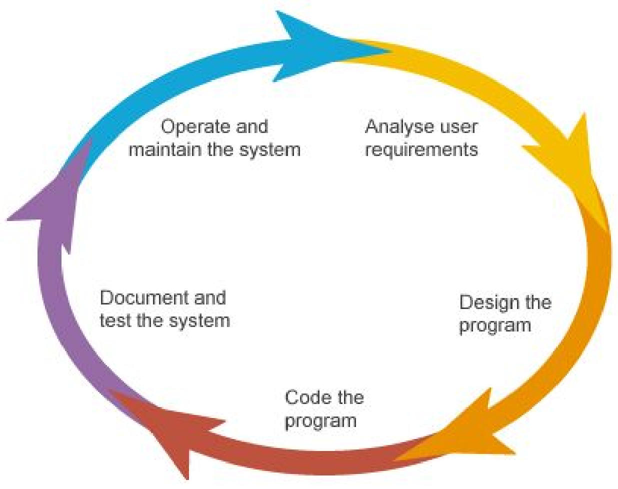

Operate and maintain the system
Analyse user requirements
Design the program
Code the program
Document and test the system

Fig 1

There are strengths and weaknesses to the above process for example it is said the this can cause increased development time and cost, there is a certain rigidity with the cycle and systems have to be outlined up front. But aside from these flaws there are many positives when it comes to this method, there is control which means large projects can be monitored, there are detailed steps to follow which can minimise confusion and error, it makes the system easy to maintain, there are standards for development and design, costs can be evaluated and there are completion targets to meet.

→ Measuring the software engineering process - Measurable data

Now that we understand the software engineering process we can look at how it can be measured when it comes to measurable data. You can measure the process in two ways, assessing data on the code itself or assessing data on the individuals and teams.

Assessing the code

One method of software measurement is metrics that are analyzed against the code itself. A software metric is a measure of software aspects which are countable or quantifiable or a measure of the degree to which a software system or process possesses a certain property. Metrics are the functions and measurements are the numbers acquired by the application of metrics. Software metrics are important when measuring software performance.

The software measurement process when evaluating code can be characterised by 5 different activities which are:

1. Formulation: Developing the appropriate metrics for the software under consideration.
2. Collection: Collecting the data so that the formulated metrics can be derived.
3. Analysis: Calculating metrics and using the mathematical tools.
4. Interpretation: Analysing the metrics to obtain insight into the quality of representation.
5. Feedback: Communicating recommendations acquired from product metrics to the software team.

Some common software development measurements are:

The number of classes and interfaces	Bugs per line of code
The number of lines of code	Code coverage
The time it takes to execute the program	Cohesion
The time it takes to load the program	Coupling
The size of the program	Comment density
	DSQI (design structure quality index)

When it comes to the number of classes and interfaces, the number of lines of code, the time it takes to execute the program, the time it takes to load the program, the size of the program it is obvious the from these metrics the data we will get back will be clear and easy to interpret as it will be measurements of these things in numbers. But we don’t actually get much information about how the code works or are they meeting the targets or demands. For example if someone has written 500 lines of code and someone has written 2000 lines of code, in a specified time, the 2000 lines could be nonsense that is an extremely inefficient way to solve the problem but it appears that they have programmed more code within the time constraints. This is why when examining the code you need to asses more than just one thing at a time. You need to use multiple metrics to gather useful information.

One of the first methods invented for systematic software testing was code coverage. Code coverage is a measure used to describe to which degree the code of a program is executed when a certain test is run.Code coverage is measured as a percentage, if a program has high code coverage it means it has had more of its source code executed during testing so it has a lower chance of containing undetected software bugs compared to a program with low code coverage. Using unit testing is a good way to achieve code coverage and if implemented properly it can identify where certain problems are in the code, for example if code coverage is at 100% and a test fails this will help you to find the specific function or part that is not working.

In software engineering, cohesion is the degree to which elements inside a module belong together and coupling refers to the degree of interdependence between software modules. Coupling is usually contrasted with cohesion. Cohesion and coupling are ordinal types of measurements and are usually described as being “high” or “low”. Low coupling is often associated with high cohesion, and vice versa. Having low coupling means a well-structured computer system with a good design, and when this is matched with high cohesion, it implies the general goals of high readability and maintainability have been reached. In my opinion having this data is a good way to help you measure the code as modules that have a high cohesion are much preferable as this associates them with multiple sought after traits including reliability, reusability, and robustness but when there is low cohesion it is difficult to maintain, test, reuse, or even understand. Therefore knowing the degree of cohesion and coupling will tell you a lot of information

Looking at the individuals and teams

When measuring the software engineering process looking at the progress from the individual or team working on the project can be extremely informative. The number lines of code, bug rates, etc., are not really good gauges of how well or poorly an individual or software team is doing, especially if they are working on very complex problems. By looking at things like number of commits, meeting targets and customer satisfaction we can gather information on the productivity and efficiency of an individual or team.

There was a study done where a group of software developers and development leaders were asked “What are the best metrics to measure software development efficiency and productivity?”. Many of them said it was an inherently difficult thing to measure. Most of them mentioned meeting targets or comparing the amount of tasks completed within a certain time frame. A method used to keep track of this is using sprints which are created with a set number of tasks and last usually two weeks. A burndown outlines the tasks to complete and how long each should roughly take this is set out at the start of the sprint. This way you can keep track of whether the tasks were completed to stay on the two week schedule.

Others said that one of the most important things when measuring the software engineering process was customer satisfaction, because what better way to measure the success of the individual or team than the customer being happy with the work. Regular check-ins to ensure that the client feels that adequate progress is being made are crucial metrics for any team. I feel that this is a good way to measure the process because at the end of the process the customer's opinion on the software is the one that matters and they decide if it was a success or a failure. It also ensures that the developers are staying true to the customers requirements throughout the process if there are regular check-ins and that they keep the first two steps of the cycle in mind, customer requirements and their initial design based on these.

→ The computational platforms available to perform this work

When it comes to platforms available for useful software analytics, that help developers understand and improve development processes and products, they started out as simple methods of collecting necessary information and data.

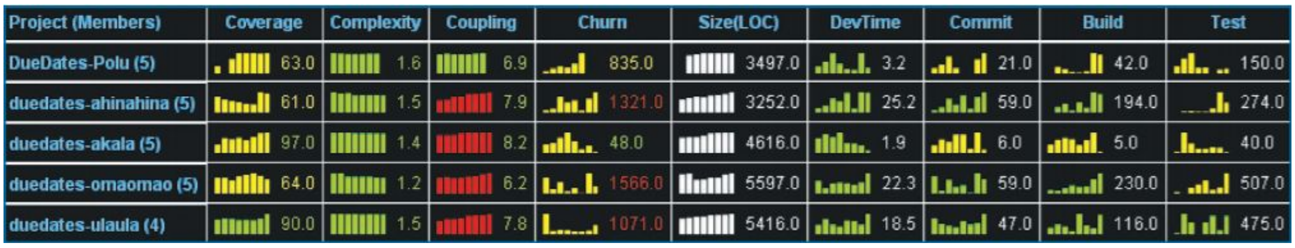
An early form of software analytics is the Personal software Process (PSP). This is a structured software development process that is supposed to help software engineers better understand and advance their performance by allowing them to track their predicted and actual development of their code. The original version of the Personal Software Process can achieve rich analytics. The PSP was created by Watt Humphreys and it was described in his book A disciple for Software Engineering. This book shows how to accustom organizational software process analytics for individual developers and PSP claims to give software engineers the process skills needed to work on a team software process. PSP uses simple spreadsheets, manual data collection, and manual analysis. A downside to PSP is that collecting and managing this data takes a substantial effort. A certain version of PSP requires developers to fill out 12 forms including:

project plan summary
time-recording log
defect-recording log
process improvement proposal
size estimation template
time estimation template
design checklist
code checklist

These forms would yield, typically, more than 500 unique values that developers would then have to manually calculate. The downfall of PSP is that the manual nature created the potential for significant data quality problems. Studies conducted involving PSP show that the manual makeup of the PSP sometimes led to incorrect process conclusions despite an overall low error rate of less than 5 percent.

Another more advanced software analytic is Leap (lightweight, empirical, anti measurement dysfunction, and portable software process measurement) toolkit. It addresses the data quality problems of the PSP. It automates and normalizes data analysis. The developer still manually enters most of the data, but the toolkit automates subsequent PSP analyses and in some cases provides analyses, such as various forms of regression, that the PSP doesn’t provide. Developers get to control their own data files. It keeps data about only the specific developer’s activities and doesn’t reference developers’ names in the data files. Leap data is also portable, this means, it creates a repository of personal data that developers can keep with them as they move around to different projects or organizations. With the introduction of automation, the Laep toolkit makes some analytics easy to collect but others more difficult.

For developers, one of the most frustrating things about manual data collection is the loop of doing some work and then having to interrupt their work to record what they worked on. A more invasive yet deeply insightful software analytic tries to deal with this problem, and it is called Hackystat, This is an open source framework for collection, analysis, visualization, interpretation, annotation, and also dissemination of software development process and product data. Developers who use Hackystat usually attach software ’sensors’ to their development tools, which will then unobtrusively collect and send “raw” data about development to a web service called Hackystat SensorBase for storage. It also has a very helpful feature called a software ICU(Intensive Care Unit) which displays the ‘health’ of a project, and shows helpful statistics based on that project.

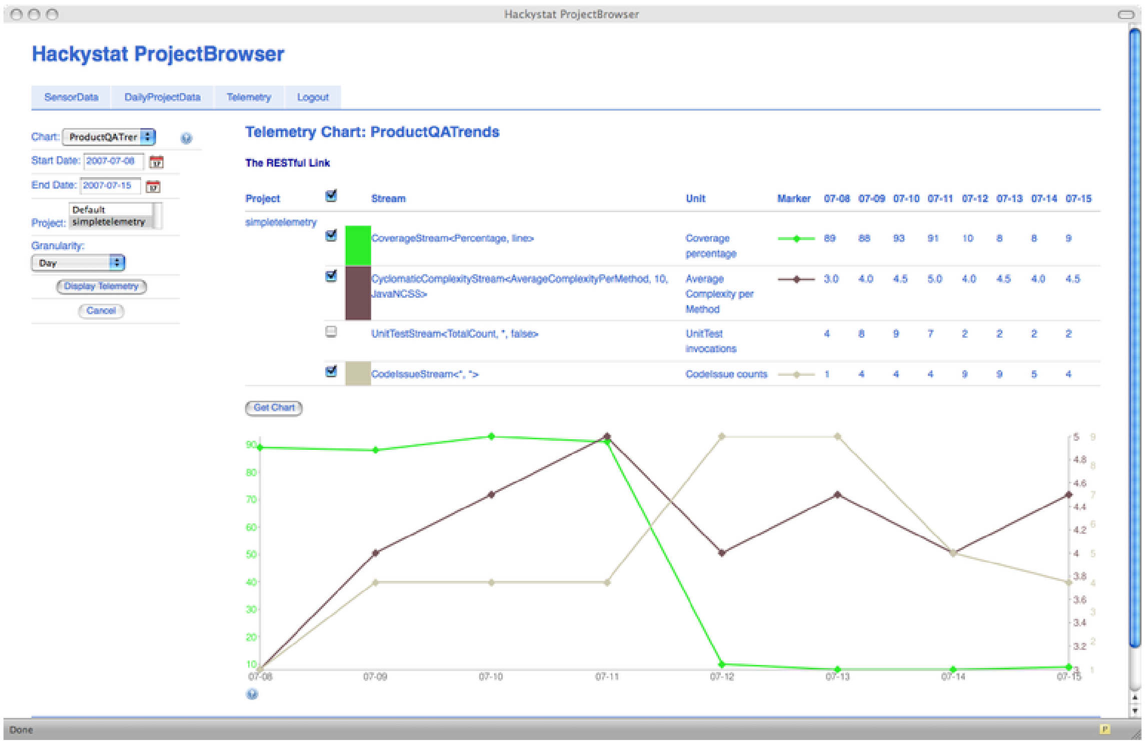


Project (Members)	Coverage	Complexity	Coupling	Churn	Size(LOC)	DevTime	Commit	Build	Test
DueDates-Polu (5)	63.0	1.6	6.9	835.0	3497.0	3.2	21.0	42.0	150.0
duedates-ahinahina (5)	61.0	1.5	7.9	1321.0	3252.0	25.2	59.0	194.0	274.0
duedates-akala (5)	97.0	1.4	8.2	48.0	4616.0	1.9	6.0	5.0	40.0
duedates-omaomao (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 2 Hackystat ICU

It provides helpful information, giving us the code coverage percentage, the complexity, the level of coupling and the churn. However on the right handside of the ICU we see some interesting data which focuses a lot on what the developer is actually doing. It gives us statistics on:

- DevTime - This determines how much time a developer spends in his or her integrated development environment working on each file associated with the project.
- Commit - This keeps track of how often each developer commits to the repository and how many lines of code he or she commits each time.
- Build - This counts how many times each developer builds the system and whether or not each build is successful.
- Test - This measures how frequently each developer invokes the test suite on the system and whether the tests ran successfully.



Hackystat ProjectBrowser

Chart: ProductQATrer
Start Date: 2007-07-08
End Date: 2007-07-15
Project: Default, simpletelemetry
Granularity: Day

Telemetry Chart: ProductQATrends

The RESTful Link

Project	✓		Stream	Unit	Marker	07-08	07-09	07-10	07-11	07-12	07-13	07-14	07-15
simpletelemetry	✓	🟩	CoverageStream<Percentage, line>	Coverage percentage	●	89	88	93	91	10	8	8	9
	✓	🟫	CyclomaticComplexityStream<AverageComplexityPerMethod, 10, JavaNCSS>	Average Complexity per Method	●	3.0	4.0	4.5	5.0	4.0	4.5	4.0	4.5
	☐		UnitTestStream<TotalCount, *, false>	UnitTest invocations		4	8	9	7	2	2	2	2
	✓	▨	CodeIssueStream<*, *>	CodeIssue counts	●	1	4	4	4	9	9	5	4

Fig 3

This image displays how Hackystat looks and how it shows its information.

In more recent years, services for software product analytics are in higher demand, with offerings from DevCreek, Ohloh, Atlassian, CAST, Parasoft, McCabe, Coverity, Sonar, and others. These services' analytics are generally built from one or more of these three basic sources:

1. A configuration management system
2. A build system
3. A defect-tracking system.

These services' analytics data focuses more on product characteristics, not the developer behaviors that produced them compared to Hackystat.

→ The algorithmic approaches available.

Artificial Intelligence(AI) is intelligence shown by machines, in comparison to natural intelligence (NI) displayed by humans and animals. The research of AI is referred to as the study of intelligent agents, which is any device that perceives its environment and takes actions that maximize its chance of success at a goal. Computational Intelligence is a subset of Artificial Intelligence. Computational Intelligence (CI) is the ability of a computer to learn a certain task from data or experimental observation. When it comes to the algorithmic approaches available concerning the data gathered and analysed on the software engineering process AI and CI are two areas which could make a noticable difference.

AI and CI are both already part of our world today and are sure to become even more important to our society in the future. It will be interesting to see how this effects software analytics. Looking at the definition of CI it is capability of a machine to learn a task from data, this means as CI evolves it could be applied to software engineering data and could help and/or change how it is collected and analysed.

CI Is a set of computational methodologies and approaches which are inspired by nature and they address complicated real life problems. They are real-world problems where mathematical or traditional modelling would be useless for a few reasons:

1. the processes might be too complex for mathematical reasoning
2. it might contain some uncertainties during the process
3. the process might simply be stochastic* in nature

*a process involving a randomly determined sequence of observations each of which is considered as a sample of one element from a probability distribution.

Many problems cannot simply be translated into binary for a computer to process it. This is where CI comes in, it provides solutions for these problems.

The approaches CI uses are similar to the human’s way of reasoning, meaning it uses incomplete and inexact knowledge when solving a problem and it can produce control actions in an adaptive way.

To achieve this CI uses a combination of five integral techniques:

1. Fuzzy logic - This allows the computer to understand natural language. It can deal with incompleteness and ignorance of data which is different to AI as it needs exact knowledge. It is useful for approximate reasoning but doesn't have the ability to learn.
2. Neural networks - Artificial neural networks which are from distributed information processing systems, enable the process and learn from data derived from experience.Neural networks have three different parts working together. The three parts process the information, send# signals and control the signals that are sent. One of the main assets of neural networks is fault tolerance.
3. Evolutionary computation - It is based off the theory of natural selection from Charles Darwin and it involves the strength of natural evolution to bring up new artificial evolutionary methodologies. It consists of other areas including evolution strategy and evolutionary algorithms. These are problem solvers.
4. Learning theory - For humans learning is bringing together cognitive, emotional and environmental effects and experiences to gather, betters or change knowledge or skills. Learning theory assists understanding how these things are processed and helps to make predictions on previous experience.
5. Probabilistic methods - This is actually one of the prominent elements of fuzzy logic. The methods evaluate the results of a CI system, greatly defined by randomness. They bring out the possible answers to a reasoning problem, based on previous knowledge.

Now if CI was applied to software analytics I feel that it could be a huge benefit. Machines could deal with huge amounts of data and using the five components above could optimize the entire process faster and better than humans ever could.

→ The ethics concerns surrounding this type of analytics

When talking about the ethics concerns involved with software analysis you always have to keep the developer in mind. Yes the client or the project manager will want to know as much information as possible when it comes to their project but there is a fine line to what type of analytics is too invasive. What information should be allowed to be gathered? How should it be collected? And who should it be available to?

When discussing computational platforms we saw that some collect uncontroversial information, this means focusing on the product characteristics. Others, like hackystat focus on developer behaviours. This is where questions about ethics come in, what type of personal data is okay to collect? In the Hackystat ICU they gathered information on the developers. They looked at the time the developer spent working on each file of the project, how often the developer commits to the repository, how many times the developer builds the project successfully and how often the developer invokes tests on the project. This is a lot of information to have about an individual and how they work, and in my opinion in some cases it could have a negative effect on some developers work. For example not everyone works or thinks the same way, some people dive straight into a project with trial and error and some people think things out thoroughly first. In the case of the latter having this type of information gathered could put them under extra pressure as they might not have as many commits or builds or time spent in the files as another developer. I feel that yes having these statistics on the developers work can be informative but if I was the developer I would feel like I was constantly being monitored.

The question when it comes to how they data is collected is whether or not it is okay to collect data without the developers knowledge. In my opinion it is not okay to keep track of a developers activities without telling them about it. They should have some say in what is tracked and also when it is shared. If a developer is constantly being monitored and all their activities are being watched that is too invasive. For example if data of what programs a worker was on on their laptops and phones each day was being tracked automatically so that the manager could see how long they were checking emails or using social media, this is too invasive a person should have the right to provide the information themselves it should not just be gathered constantly and automatically, this is too much pressure.

Who the information, gathered on a developer, is open to is important. In a work environment if statistics are open to everyone this would create a certain competitive environment and even though this may suit some people you have to take everyone into account. Comparing people against their peers is natural in any job this is how management can make decisions on staff, but allowing this information to be available to everyone could hinder some people's performance. The developer should definitely be aware of who is allowed to access the data collected in regards to their work.

→ In conclusion

In my opinion this essay title was an interesting one to research. I enjoyed reading the opinions of real software developers and development leaders. When it comes to measuring the process I think looking at things like customers satisfaction and meeting targets is the most informative and ethical. With data like this you aren't putting a single developer under a microscope but more looking at the picture as a whole. I would certainly enjoy working in an environment that doesn't scrutinize every aspect of a developer but looks more at the data on the product characteristics better than an environment that focuses on the individual developer behaviours.

→ Sources

1. https://en.wikipedia.org
2. https://stackify.com/measuring-software-development-productivity/
3. http://www.citeulike.org/group/3370/article/12458067
4. http://csdl.ics.hawaii.edu/research/hackystat/