13/11/18

Software Engineering (CS3021)
Colleen McGloin
SN:16325155

"Deliver a report that considers the ways in which the software engineering process can be measured and assessed in terms of measurable data, an overview of the computational platforms available to perform this work, the algorithmic approach's available, and the ethics concerns surrounding this kind of analytics."

Introduction:

"If you can't measure it, you can't improve it."[1] Measuring the properties of an entity can result in useful statistics and information. We can use this knowledge to improve the quality and reliability of our object of interest. We measure Software Engineering in order to gather practical and quantifiable data so we can use these measurements for important necessities in project's such as budget planning, cost estimation and software development.[2] There are many benefits to these measurements such as ascertaining the quality of the ongoing process and also using the collected data to find areas of improvement. This can lead to better productivity throughout the Software Development Life Cycle (SDLC). There are different types of data that we can gather from measuring Software Engineering by using Computational Platforms like "Hacky Stat" and "The Leap toolkit"; all of which I will review in this report. Lastly, not everyone agrees that software metrics deliver the value and benefits its proponents suggest, which I also explore later on in this report.

Measurable Data:

Software measurements can be used for different reasons by different people on a team such as the Project Manager or the programmer. A Project Manager can use software measurements in the development process to measure aspects such as team productivity, software efficiency and project scheduling. All of these attributes are measured in order to keep control of costs, the delivery schedule, project resourcing and also to find areas of the software process that needs enhancement. Lines of Code, Cyclomatic complexity algorithm, Function Points and Code Churn are some of the most popular types of metrics used in the workplace.

Lines of Code:

Lines of Code (LOC or KLOC which stands for thousands of lines of code) has been a consistently used metric in measurement of Software Engineering. It is used to calculate a program's size and can be extrapolated to show a programmer's productivity and program quality. Lines of Code measurements are used in the basic model of COCOMO (The Constructive Cost Model). This basic procedural model is used in projects to calculate a

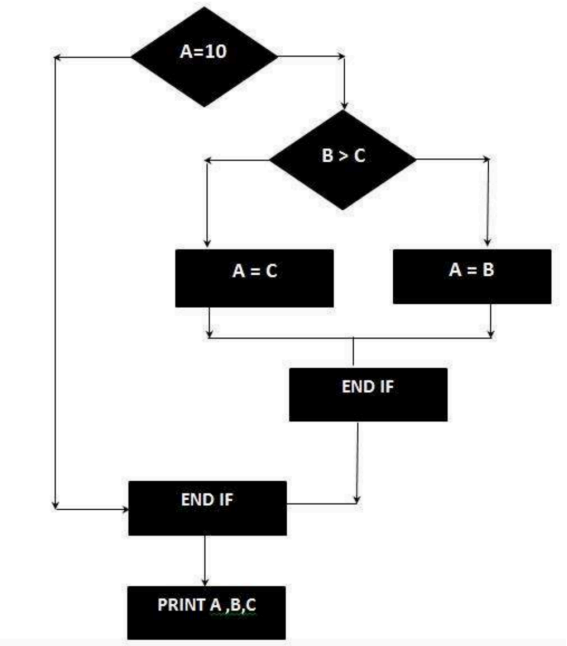
rough estimate of effort that a person puts in. It also can estimate the cost of projects along with schedule and size.[3,4]

One of the down falls to using Lines of Code data is that a programmer can count the lines in 2 different ways. The first way is counting physical lines of code. The issue here is that some lines of code could contain comments or dead code. The second way is counting just logical lines of code in the program. The main issue arises when analysing this data and comparing software due to the fact that some programs LOC may have been counted using the first method while others could have used the second method of counting.[5]

It is evident that LOC is a very harsh measure of a programmer's effort as many modern programming languages specialize in producing clean code in as little lines as possible. Despite this harshness, LOC measurement still remains a very popular metric. It is used widely because of its simple calculation nature and to give an estimation of a programmer's output. (For example, calculating LOC per programmer per month).[6]

Cyclomatic Complexity:

Unlike the LOC measurement that is used to calculate a programs size and programmer's effort in a project, Cyclomatic Complexity algorithm is used to measure a program's complexity and quality. It was developed in 1976 and still remains to be a very popular software measurement. It uses a control flow graph relating to the code being measured and assesses the amount of linearly-independent paths in the program.



The nodes in the graph represent each condition present in the code. The edges in the graph represent the different paths that the code can take when the program pointer hits the condition. The Cyclomatic Complexity is calculated by using the following equation:

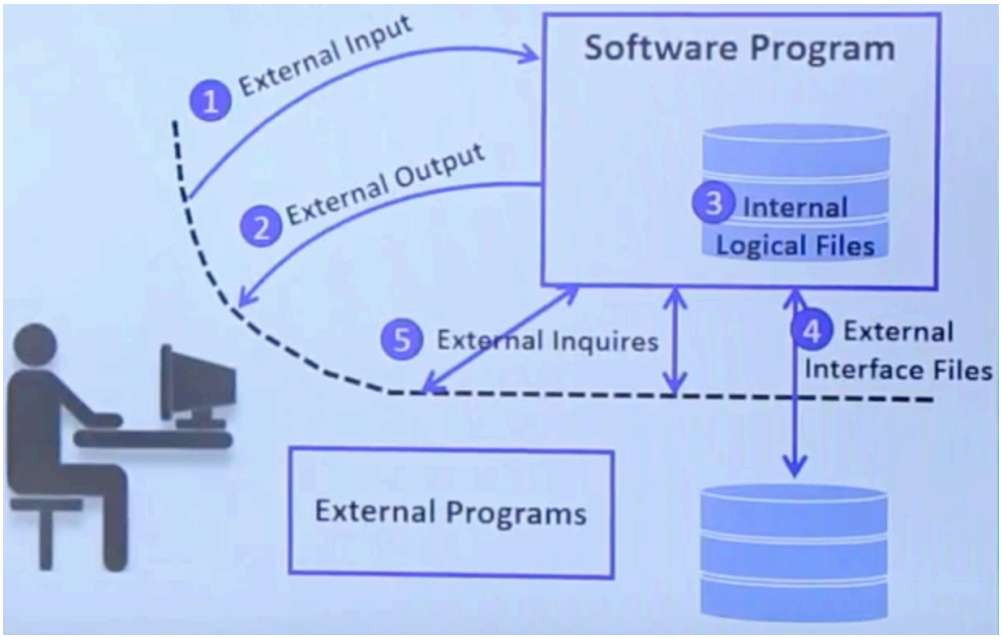
$$Cyclomatic\ Complexity = E - N + 2 * P$$

where E represents number of edges in the graph, N represents the number of Nodes and P represents the amount of nodes that have an end point. In this example, the Cyclomatic Complexity is 8 – 7 + 2* 1= 3.

The lower the complexity, the easier the program is to comprehend and the less complicated the code is. The company Hewlett Packard considers a function in a program having a Cyclomatic complexity greater than 16 to be redundant and the programmers must redesign the function.[7]

Function Point:

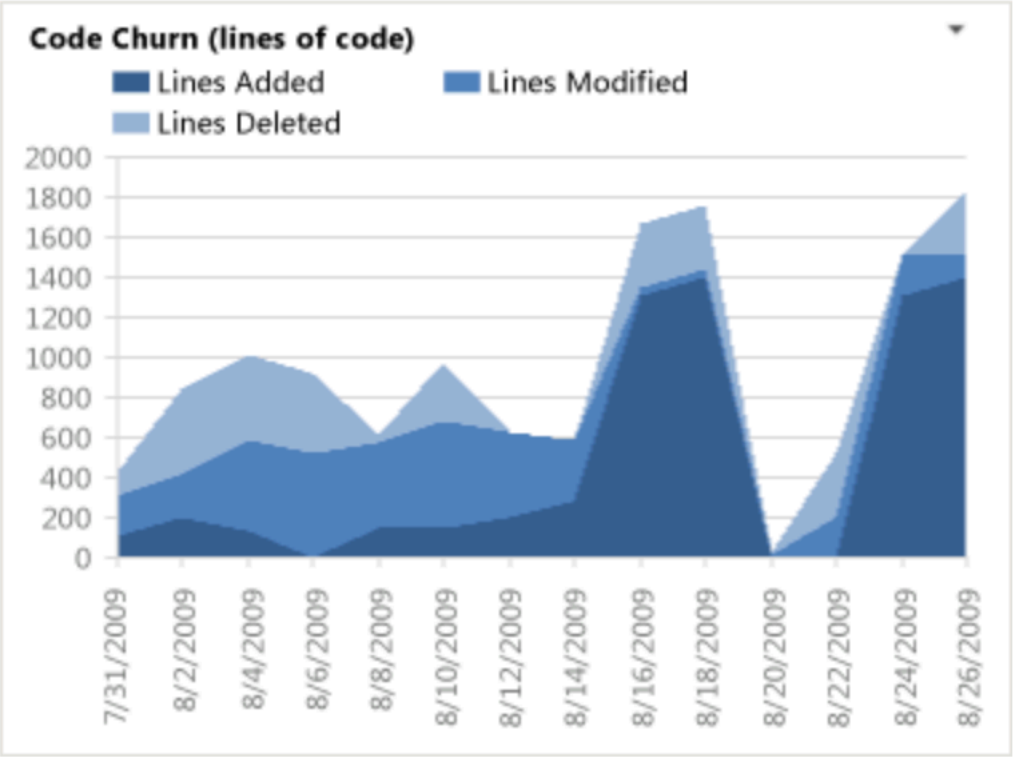
Function point measurement has been used as an alternative to the LOC software metric in recent years in an effort to overcome the limitations of LOC. Function Points are a unit of measurement that represents how much business functionality the program delivers to the end-user. The greater the amount of function points, the more functionality a program has. An advantage to function point measurements is that is can be calculated at any stage in the SDLC. In order to calculate the function point number, one must divide each functional user requirement into 5 categories: External inputs, External outputs, Internal logical files, External interfaces and External inquires. This can be done multiple times during the development process and can be used to benchmark a developments progress against project plans.



Upon completion, one decides on each requirement's type of complexity (either simple, complex or average) and then the requirement is assigned a number of function points based on this complexity by using a standardized set of basic criteria. These individual function points are totalled and then normalized by using other characteristics relating to the software as a whole. We are left with a final integer value which is known as the Function Point Index and we can use this index to measure program size, cost per unit of a piece of software and the productivity of the software process.[8]

Code Churn:

Code Churn is a software measurement that allows software engineers to see the rate at which code on the project is changing. It is the sum of lines added, deleted and modified over a period of time. It is a good indicator to software managers on how their team is performing by seeing who on the team is writing more useful code and who is rewriting their own code frequently. Having a significant high or low level of code churn could show that the functional requirements are unclear to the team members or else the project spec is constantly changing. It is quite common to see a high level of code churn at the start of the project as programmers are being adventurous in their ideas and testing out different solutions.[9,10]



Screenshot taken from Stackify, it is an example of a code churn report produced by visual studios.

By periodically analysing code churn during the development stage, the team will be better informed as to whether additional resources are needed, project delivery dates need to be extended or functionality needs to be dropped from current release. Early sight of potential delivery issues allows for appropriate intervention by senior stake holders.

Computational Platform:

There are multiple platforms that both software engineers and software development managers can use to collect and analyse software metrics such as PSP, Hackystat and the Leap Toolkit.

PSP:

PSP (Personal Software Process) is the original methodology used by software engineers to track and record their progress on projects. The objective was to improve their project performance and ensure alignment to key project milestones and deliverables.[11] It is also used to improve program quality. Types of data that are gathered for the PSP include: project size, number of defects contained in the program and program run times. All this information is then analysed and then used to make estimations on time schedule, programs condition and also to make known problematic areas in the project. By being aware of the issues in the program early on in the SDLC, it can lead to early intervention to address issues and improve team productivity. Unfortunately, there are disadvantages to the PSP. Developers collect data from their program and manually record the data on paper. In one version of the PSP, engineers were required to fill out 12 forms which was extremely time consuming and demoralizing.[12]

Here is an example of a defect log that was filled out by an engineer using the PSP methodology:

Name: Jill Fonson **Program:** Analyze.java

Date	No.	Type	Inject	Remove	Fix time	Fix defect no.	Description
9/2	1	50	Code	Com	1	1	Forgot import
9/3	2	20	Code	Com	1	2	Forgot ;
9/3	3	80	Code	Com	1	3	Void in constructor

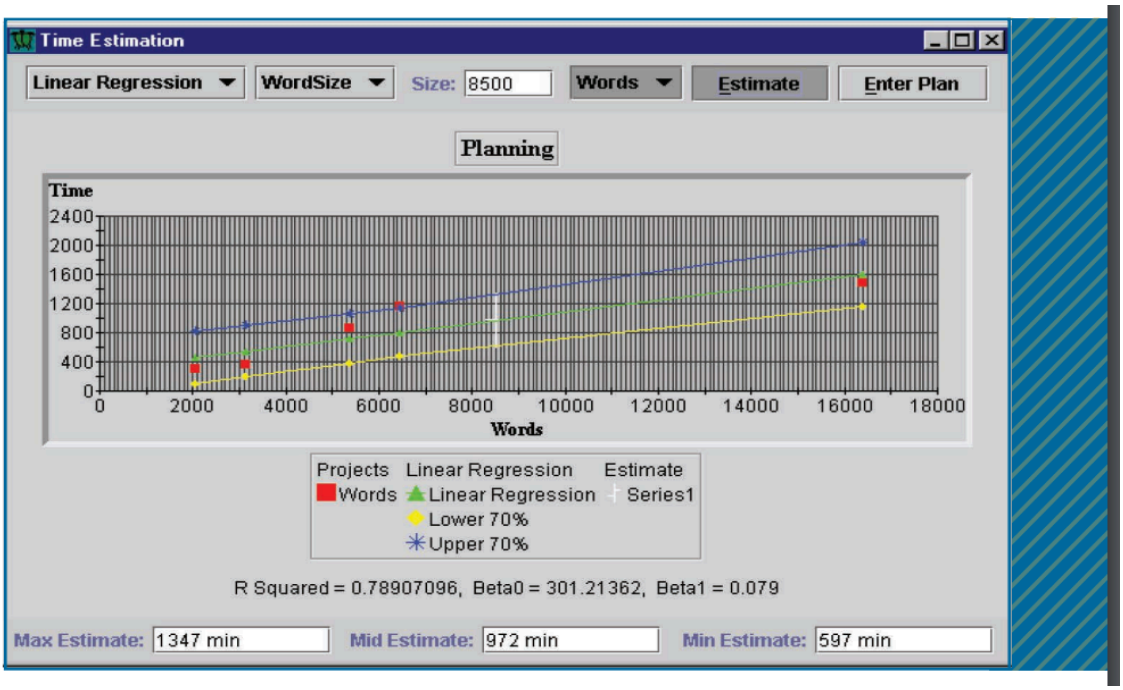
Screenshot taken from “Searching under the Streetlight for Useful Software Analytics” paper.

As you can see, the level of detail required for the PSP is very extreme. Syntax errors are recorded which evidently makes the process of data collection long. In a study conducted by a Software Engineering research team, 72 out of 78 students that were taught PSP “abandoned” the methodology as they believed the extra work required to gather the data “would not pay off”.[13] Efforts were then made to create a similar methodology to the PSP that involved the automation of data collection. This led to the development of the Leap Toolkit which allowed for detailed analysis of the data collected.

Leap Toolkit:

The Leap Toolkit was designed to combat the issues that began to arise with the PSP. Considerable data quality errors were beginning to pop up and it is believed to be because users had to input data manually on paper for the PSP. To tackle these quality errors in the PSP, the Leap toolkit was created. Developers can record many things using the toolkit such as the scale of the project the developer is working on, the time it takes to produce the project, errors that engineers find, checklists that are being used in the process, project goals and many more aspects of the software engineering process. Even though the user is manually entering in data, similar to the PSP, the LEAP toolkit allows for further PSP analysis and also provides additional PSP findings. After the user inputs the data, the Leap Toolkit automates this PSP analysis and then can issue the new PSP data. In comparison to Hackystat, the toolkit only stores information relating to the individual developer. It stores the data in a portable repository that the developer can continue to use while working on new projects.[14,15]

Here is an example of the Time Estimation component of the Leap Toolkit:



Screenshot taken from "Searching under the Streetlight for Useful Software Analytics" paper.

From the above screenshot, you can see that the higher the predicted word count in the project, the greater the estimate for time to completion.

Users of the Leap Toolkit began to realise that the platform was not very beneficial for context switching i.e. switching from gathering data related to product development and process documentation. As a result, Hackystat was born.

Hackystat:
Hackystat is a framework that is available for many different types of software developers. It allows users to either gather and analyse data in relation to their current software product/process, evaluate the users own PSP data and track their ongoing progress in the software development or it can introduce Software Engineering students to software metrics.

Hackystat works by users attaching sensors to their product's development tools and these sensors, unobtrusively, gather and deliver data to a server called the "Hackystat SensorBase". By having the data on a server, it is easily accessed by other web services which can then inspect the process/products data at a higher level of analyzation. These web services can generate visual representations and/or annotations of the data. There are many advantages to Hackstat's design such as its ability to collect data from both the user's local desktop and the server the user is using to store data. It collects the data in a way that the software engineer isn't aware of. For example, Hackystat stores data when the user is working offline and then once the user is reconnected, the data is uploaded to the SensorBase server. It can draw data minute by minute or even second by second and it also can collect information while the user is actively editing the code![16]

Here is an example of Hackystat's gathered data from an ongoing software development process:

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

Screenshot taken from "Searching under the Streetlight for Useful Software Analytics" paper.

It is evident that various types of informative data relating to all team members is recorded which can subsequently be used by the programmer themselves or by the management team.

Ethics:

Many software engineers stand by software measurement and believe it is a very effective way to track the progress of an engineer or project and it also allows developers to value their work more. Many IT companies and government agencies such as NASA and the US military are firm believers in software measurement. However, there are some people that are quite against the use of software metrics for many reasons. Some of these reasons include the ethical concerns surrounding the analysation process and also the fact that some believe it impacts developers negatively as one becomes more stressed which leads to bad performance levels in the project.[17]

Privacy Concerns:

There is a general opinion amongst groups of software developers that software measurement lacks basic privacy control for individuals. They argue that for the amount of data that is being analysed and stored on team members, there is very little privacy regulations taken to protect these team member's data. For example, Hackystat automatically collects copious amounts of detailed information about the developers and they began to worry that this data may be accessed by senior management teams. However, Hackystat took these concerns on board and implemented multiple steps to protect the developer's data. These issues were addressed by password encryption on data and allowing developers to download the SensorBase server so only he/she could access it on their local machine.[18]

Issues with measurable data being meaningful:

As previously discussed, there is one major issue associated with the LOC metric for software measurement; when it is used to assess a developer's effort in a project. Before there were programming languages that enabled developers to produce large scale programs in as minimal lines as possible, the higher the amount of lines in a program showed how much time and effort a programmer put into the project. However, in the fast paced software development world we live in today, it is virtually impossible to judge an engineer's effort on the amount of lines of the code he/she coded in a program. It is said that the coding elements of a project only amounts to 30% of the overall effort given to a project.[19]This raises the question of whether the LOC metric should still be used as a software measurement for analysing a programmer's effort in software companies. By continuing to use this measurement as a way to assess effort, it will result in programs that are unnecessarily long which will in turn lead to more costs and evidently more bugs in the program.

Is there a point to software metrics?

The software measurement process is an extremely valuable tool to use during the SDLC of a project but when deadlines are approaching and stress is high in the workplace, this process of software measurement starts to slow down. Developers find it very difficult to keep on top of this measurement activity while simultaneously finishing projects. It is also common for developers to measure the wrong things in a program which results in non-

beneficial information at the end of the project. The correct measurements are taken when the developers ask themselves does the data being collected relate to the overall purpose of the project and also will management teams be able to extract the necessary information from the data collected.[20]

Conclusion:

Evidently there are multiple ways that the Software Engineering process can be measured from Lines of Code to Code churn using multiple platforms like Hackystat and the Leap Toolkit. As we have seen, software metrics is constantly evolving, just like software engineering. Some may argue that software metrics is useless and a waste of time while others rely on it to produce their products successfully. Personally, I believe software metrics is a great tool that allows developers to keep track and value their progress over time.

References:

1. https://www.growthink.com/content/two-most-important-quotes-business
2. https://stackify.com/track-software-metrics/
3. https://www.geeksforgeeks.org/software-engineering-cocomo-model/
4. https://en.wikipedia.org/wiki/COCOMO
5. https://stackify.com/track-software-metrics/
6. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.63.2683&rep=rep1&type=pdf
7. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.63.2683&rep=rep1&type=pdf
8. http://www.ifpug.org/about-function-point-analysis/
9. https://blog.gitprime.com/why-code-churn-matters/
10. https://blog.gitprime.com/6-causes-of-code-churn-and-what-you-should-do-about-them/
11. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.138.6806&rep=rep1&type=pdf
12. http://www.citeulike.org/group/3370/article/12458067
13. https://www.researchgate.net/publication/4016775_Beyond_the_Personal_Software_Process_Metrics_collection_and_analysis_for_the_differently_disciplined/references
14. https://www.researchgate.net/publication/221554968_Lessons_learned_from_teaching_reflective_software_engineering_using_the_Leap_toolkit
15. http://www.citeulike.org/group/3370/article/12458067
16. http://www.citeulike.org/group/3370/article/12458067
17. https://en.wikipedia.org/wiki/Software_metric
18. https://www.researchgate.net/publication/4016775_Beyond_the_Personal_Software_Process_Metrics_collection_and_analysis_for_the_differently_disciplined
19. https://www.ppi-int.com/wp-content/uploads/2018/02/The-Mess-of-Software-Metrics_Jones-C_2017.pdf
20. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.138.6806&rep=rep1&type=pdf