Measuring Software Engineering

Name: Imoesi Idogho

Student No: 18318386

Date: 25/11/2020

Software Engineering as a field and faculty is one that, without a doubt, a rapidly evolving and changing industry. New processes, methods and approaches delivering increase efficiency and/or promoting greater feature/power in addition to the developments of new technologies entirely lend to ever-constant need to quantify and measure various factors such as the reliability and stability of work produced or deal with the quantifications of test results on said work so on and so forth.

This report deals with the segment of engineering tools used for such processes: that is those that measure working performance and the computational platforms available for such and the effects and pressures the use of these such tools places on the software engineering process and the individual software engineer themselves.

Software measurement

Software measurement is oft the name given to the above segment of tools and at the barebones level it has possesses to main points of discussion for each engineering project. Such being what data is to be collected and how is that data to be processed and presented in order to glean useful information from it. Such can focus purely on the code while there are others that can glean and analyse other more external areas of data; to take a section of these measurements we have

Lines of Code

Also, known as source lines of code (SLOC), it is a software metric that is used to measure the size of a computer program by counting the number of text lines in the program’s source code. SLOC is mostly used as an indicator for programmers to see how much work to be done and is left to do for the program to be complete as well as to estimate programming productivity or maintainability once the software is produced.

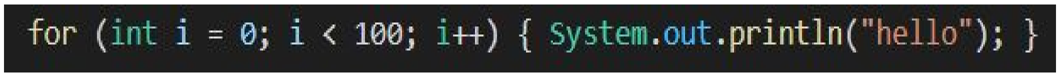
There are two primary types of SLOC measures:

a) Physical Lines of Code (LOC) - Which is a count of lines in the text of the program’s source code excluding comment lines.

b) Logical Lines of Code (LLOC) - This measures the number of “statements” however their specific definitions are related to specific computer languages i.e. a simple logical LLOC measure for C-like programming languages is the number of statement-terminating semicolons.

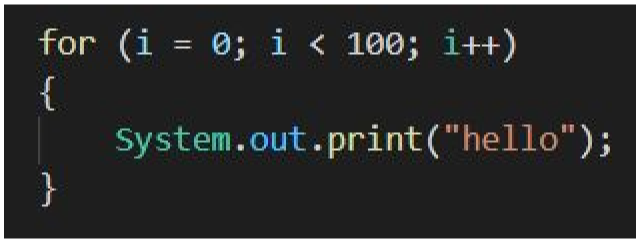
It’s much easier to create tools that measure physical SLOC. Furthermore, physical SLOC definitions are easier to explain that Logical SLOC. However, physical SLOC measures are sensitive to logically irrelevant formatting and style conventions, while logical SLOC is less sensitive to formatting and style conventions.

Consider the two following code snippets -



```
for (int i = 0; i < 100; i++) { System.out.println("hello"); }
```

Lines of Code (LOC)



```
for (i = 0; i < 100; i++)
{
    System.out.print("hello");
}
```

Logical Lines of Code (LLOC)

Both are functionally the same yet the first has 1 LOC and the second up to 4 depending on the counting of brackets. Depending on how one discards brackets and even large amounts of whitespace such questions as the values of non-statement lines, bracing work, or even the value of comments, come into being.

Bugs per Line

Another method used to measure code quality is by measuring how many errors or bugs exist in an average line of code. Usually measured in errors per n lines of code such allows an image of the quality of the software being written, in addition to a measurement of the ability of the engineers dealing with such.

The bugs per line can also be utilised in measuring work rates as during periods of bug quashing the fixing of said errors can be tallied and assigned per engineer. However, not all bugs are equal in scope and ability and as such care must be taken as in such an unassigned variable can occupy the same space of value as a broken or inaccessible code system.

Balance Scorecard

“A balanced scorecard is a strategy performance management tool… that can be used by managers to keep track of the execution of activities by the staff within their control and to monitor the consequences arising from these actions.” In focus, the balance scorecard refers to a performance management report used by a management team in regards to the team’s implementation of whatever project or strategy goal is being worked towards.

Two of the ideas that underpin modern balanced scorecard designs concern making it easier to select which data to observe, and ensuring that the choice of data is consistent with the ability of the observer to intervene. However, one of the main criticisms of the scorecard is its failure to provide a bottom-line score or a unified view with clear recommendations: it is in many ways simply a list of metrics.

As we see in general, one is trying to measure the overall software productivity of both the engineer and the work in question. These can see a general expansion into external metrics like time on a project or even engineer work satisfaction that can be used to measure speed or work done and also are part of the measurement suite.

Computational Platforms

Whatever measurement or metric has been decided to be taken up and data decided to be measured said data must be gathered, classified and analyzed; to make it, easier to extract conclusions and strategies out of it.

This could be called Software Intelligence; the same way Business Intelligence "offers concepts and techniques to improve business decision making by using fact-based support systems, SI would offer software practitioners (not just developers) up-to-date and pertinent information to support their daily decision-making processes."

These systems range from the totally manual to the totally automatic and can range from dedicated toolsets to integrated interfaces within a wider system. The choice to be made on which platform is very much of which metrics are being measured and are relevant to that of the projects possible overhead financially and technically. The platforms themselves include -

PSP and LEAP

PSP was created to help software engineers better understand and improve their performance by bringing discipline to the way they develop software and keeping track of their predicted and actual development of their source code. It occupies part of the manual range of platforms heavily relying on manual data input and analysis. In total twelve possible forms to be filled out, with 500 distinct possible values.

Watts Humphrey created this platform 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" claiming to give software engineers the process skills necessary to work on a team software process (TSP) team.

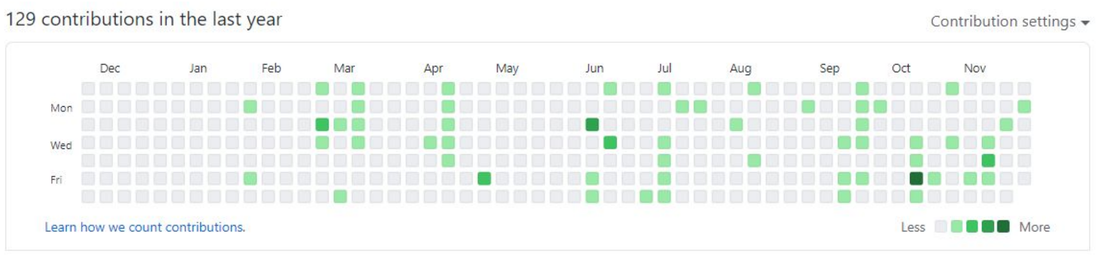
The LEAP toolkit descends from the Collaborative Software Development Laboratory (CSDL) from when the University of Hawaii set out to design their own piece of software led by C. Moore, the LEAP toolkit after using PSP for two years. The goal of the LEAP (Lightweight, Empirical, Anti-measurement dysfunction, and Portable) toolkit was to address data quality problems with the PSP arising from human error during manual data collection and analysis.

LEAP encompasses the core of the necessities around computational platforms with its two main activities of gathering data and performing analyses on such in additions to other processes like retooling data definition and ability on individual engineer skill acquisition and measurement.

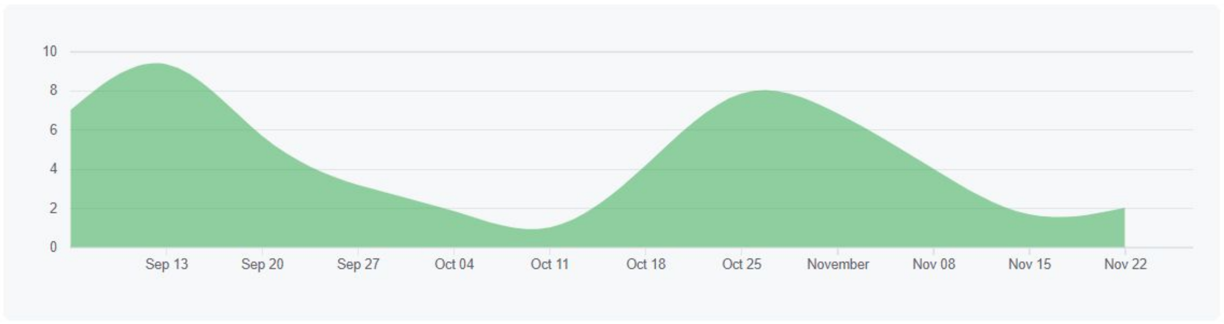
Repository Platforms

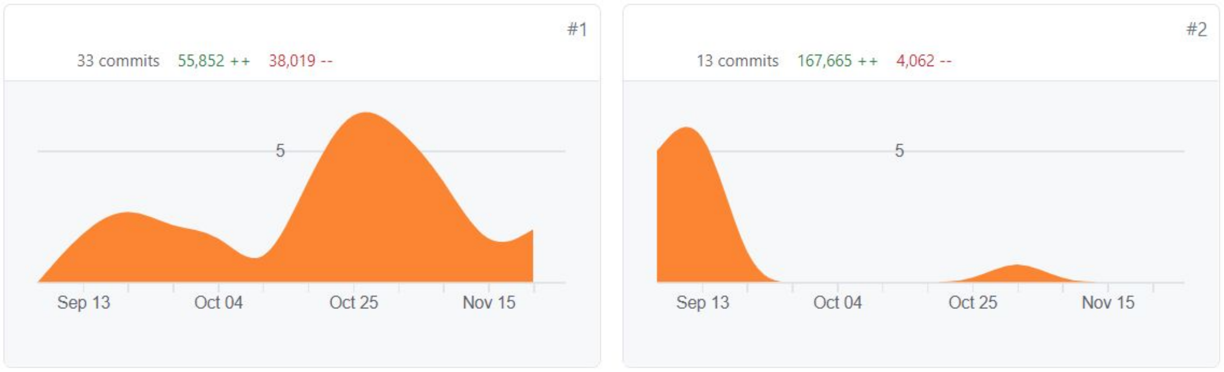
Platforms like the various Git interfaces or Atlassian's Bitbucket often have their own platforms integrated into the wider system that collect, deliver and visualise pertinent information and present it to be acted upon. Such systems like Git Prime (Pluralsight) or Jira can differ widely in what data is collected and the level automation varies from service to service but often said systems can be fined tuned from team to team and project to project depending on the metrics focused on.

Generally what is dealt with is the fronting of statistic visualisation which allow a team to measure various metrics most often dealing with work amount by size or frequency but also other metrics. Well know visualisations from these including examples such as the developer's punchcard

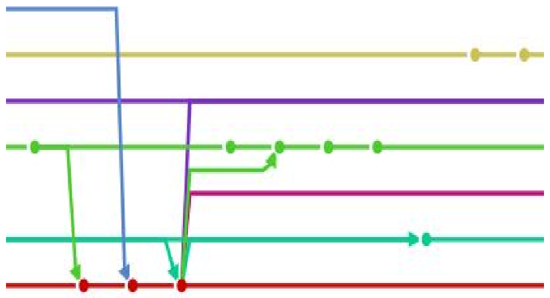


Or commit measurements





And even branch and fork interactions



Algorithmic Approaches

As always maths and algorithms infect everything and as such the field of software development has seen great growth with several algorithms designed to facilitate and aid the software development process. Often utilising data gleaned through measurement and presented by the computational processes these processes can be oblique in their actual value and will often differ in effectiveness and usefulness from project to project and team to team.

Machine Learning

“Machine learning is an application of artificial intelligence (AI) that provides systems with the ability to automatically learn and improve from experience without being explicitly programmed.”

Machine learning focuses on getting the computer program to be able to grow and learn from happened experiences and positive feedback either external or not rather than from through programmed instructions that tell the program what to do.

On the whole machine learning can be used to processed gathered data and deliver useful results and recommendations in line with its learned predictions and recommendations making it a powerful tool in preparing, managing, predicting and analysing the weight and expectation of various metrics from and involved bodies such as the individual engineer or general team productivity so on and so forth.

Machine Learning can be divided into 4 types of algorithms:

1) Supervised Learning
2) Unsupervised Learning
3) Semi-Supervised Learning
4) Reinforcement Learning

Supervised Learning

Supervised learning is where you have input variables (x) and an output variable (Y) and you use an algorithm to learn the mapping function from the input to the output.

$$f(x) = y$$

The machine is to be taught the mapping function so well that when you have new input data (x) that you can predict the output variables (y) for that data without the actual y with a high degree of confidence. As such sounds large amounts of data are needed to properly set up the algorithm with possibly a large overhead of cost and the data metrics involved can be quite constrained. That said supervised learning is the most common of practical machine learning.

An example of learning problems from such:-

- **Classification**: A classification problem is when the output variable is a category, such as “red” or “blue” or “disease” and “no disease”.
- **Regression**: When the output variable is a real value, such as “dollars” or “weight”.

Unsupervised Learning

“Unsupervised learning is where you only have input data (X) and no corresponding output variables. The goal of unsupervised learning is to model the underlying structure or distribution in the data in order to learn more about the data.”

Supervised learning systems generally start with a known data training set from which the system makes in inferences about output values. With sufficient training, new data passed in will be correctly targetted and passed out. On the whole “The system doesn’t figure out the right output, but it explores the data and can draw inferences from datasets to describe hidden structures from unlabeled data.”

An example of learning problems from such:-

- **Clustering**: In which you process for the inherent groupings in the data, such as grouping customers by purchasing behaviour.
- **Association**: Which is where you want to discover rules that describe large portions of your data, such as people that buy X also tend to buy Y.

Semi-Supervised Learning

Semi-Supervised learning straddles the middle way, as its name suggests, generally dealing with large amounts of unlabeled data (unknown Y’s) and lesser amounts of labelled data (known Y’s). A good example is a photo archive where only some of the images are labelled, (e.g. dog, cat, person) and the majority are unlabeled such as that used and trained on the Google Captchas like some of the systems used to train self-driving cars.

“Usually, semi-supervised learning is chosen when the acquired labelled data requires skilled and relevant resources to train it / learn from it. Otherwise, acquiring unlabeled data generally doesn’t require additional resources.”

Reinforcement Learning

Reinforcement Learning is a learning method that interacts with its environment by producing actions and discovers errors or rewards. As the output is delivered the system is fed said errors or rewards based on whatever feedback from the environment that is delivered.

“Trial and error search and delayed reward are the most relevant characteristics of reinforcement learning” and if done properly this method allows machines and software agents to automatically determine the ideal behaviour within it’s given program niche and context in order to maximize its performance i.e. the golden mean.

On the whole, for reinforcement learning, less time is usually spent designing the model as there are no fast rules other than reward wanted behaviour. Allowing for a “low-bar” into working with said models as long as the system is understood.

Ethics

All of these measurements and compilations in regards to data and the people they come from do raise several ethical concerns. From the collection of said data in the first place to questions about its use to the questions of the correctness of said data in the first place and reflecting such onto the real world.

Privacy

The inherent danger behind the mass collection of data and metrics is one of its improper use and improper access. Personal data, by far the most sensitive, though seemingly somewhat away from the rigours of the development process can pertain in regards to things like employee happiness, or health and personal concerns that may affect ones working life (e.g. pregnancy or injury).

The collection of said data can present active danger or promote possible distress if misused or mishandled (such as firing a pregnant woman to prevent “downtime”) in addition to the ever-present danger of data leaks. Such also extends into the long-term retention of data on non-present subjects and the ensurance of strong safeguards and ethical oversight on data usage.

Accuracy

Another point of ethical concern can be the accuracy of said data and the dangers that emanate from the application of non-accurate data onto systems and people. Data can be made non-accurate deliberately (such as someone inputting altered numbers into a manual computational platform) or non-deliberately for reasons from present knowledge that data is being collected to quiet nuances in the subject or systems measured e.g. a system that takes the measurement of worker effort in LOC and notices drop-offs in spring and as such leads to approaches to combat the drop in productivity by the data's analysers ignoring the innate possibility of the occurrence of seasonal illnesses that may temporarily weaken the working group.

These are a small cross-section of the ethical concerns that must be taken on-board as one sets up a software measurement schema in the search for greater efficiency and productivity that are to cause necessary stops in raising questions of can, should, shall and so on.