**MEASURING SOFTWARE ENGINEERING**

Muhammad Rvail Naveed

Student No: 1732983

**Introduction**

Software Engineering can be defined as the systematic application of scientific and technological knowledge, methods, and experience to the design, implementation, testing, and documentation of software. Modern software is becoming increasingly more and more complex at an exponential rate. Since it’s inception in the late 50’s it has involved into a profession concerned with how best to maximize the quality of software and of how to create it. With technology advancing so fast, the need to understand how to make the engineering process faster and more efficient has advanced with it. Over the years there have been many discussions on how best to measure the effectiveness of the software engineering process and the productivity/competence of the developers responsible for constructing the products.

There are many metrics and methodologies that can be used to determine this. This report will discuss some of these widespread methods, their usefulness, benefits and possible ethical concerns that may arise from the collection of such data.

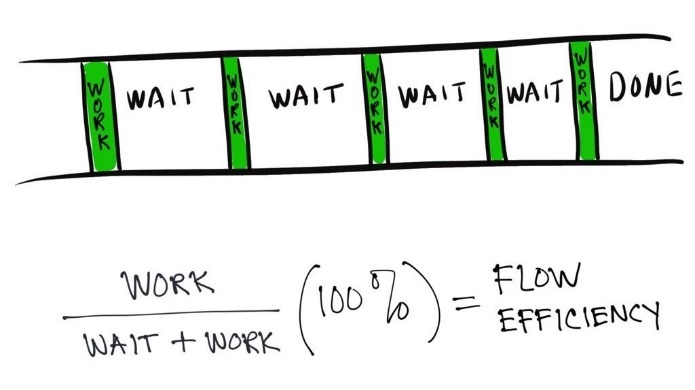
**Measuring and Gathering Metrics**

One of the most basic metrics that can be measured is Source lines of code (SLOC). It is used to measure the size of a program by counting the number of lines in the source. It is used to predict the effort it will take to produce a piece of software and can also be used to estimate programmer productivity once the software is complete. The basic approach is “the better the programmer, the more lines of code they will have written”. There are two main ways of measuring SLOC commonly referred to as physical SLOC and logical lines of code (LLOC). In physical SLOC the method is to count every line of code that the programmer has written, excluding comments whereas LLOC tries to determine how many statements are in the source code. Both ways can be achieved relatively easily by simple algorithms such as “wc” from GNU’s Coreutils for SLOC.

Some advantages of using SLOC as a metric for measuring the software engineering process is that it is easy to automate the process of counting the lines and many utilities to do this already exist. Another advantage is that it is an intuitive metric, it can be seen and the effect of it can be visualized. However, the bad greatly outweighs the good. As someone once said “using SLOC to measure the software progress is like using kg for measuring progress on aircraft manufacturing”. The SLOC metric encourages bad practises such as copy-paste syndrome and discouragement of refactoring to make things easier. Just because a developer is producing 10,000 lines of code a week does not mean that that code is optimised or even completely bug free. The lines of code would also depend on the experience of the developer, number of lines differs from person to person and an experienced developer may implement certain functionality in fewer lines of code than another developer of relatively less experience, even though they use the same language.

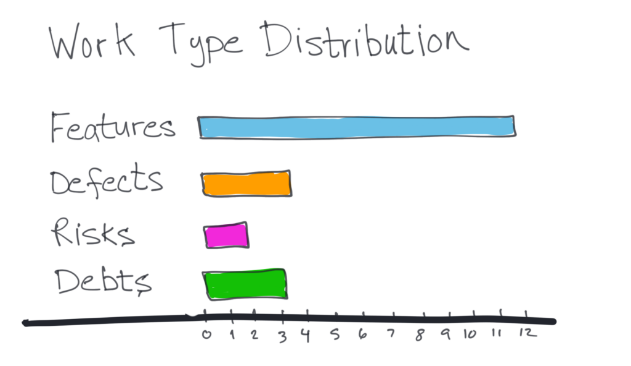
Hours spent programming is another very basic metric that by itself is not very useful. Although it can give a quick insight as to how long a specific problem is taking or how long it took to deploy a new feature. It is better integrated alongside Test Driven Development(TDD). Research has shown that doing TDD greatly reduces the amount of other non-coding work that developers must do, and this in turn motivates developers to get on board. A 2005 study, [*"On the Effectiveness of Test-first Approach to Programming"*](http://nparc.cisti-icist.nrc-cnrc.gc.ca/npsi/ctrl?action=shwart&index=an&req=5763742&lang=en), found that using TDD meant writing more tests and, in turn, programmers who wrote more tests tended to be more productive.

Flow metrics offer a collection of very useful tools to help identify many things such as how long a piece of software is taking to complete and efficiency of development. “Flow” can be thought of as value that can be pulled through a system smoothly and predictably. Flow time is the amount of time a flow unit spends in the development pipeline from conceptualisation to deployment. This can be useful to help the team understand which items are taking longer to complete so more resources can be dedicated to that task. Flow efficiency examines the two basic components that make up flow time: working time and waiting time. Waiting time can be encountered for many reasons such as: dependencies, priority changes, too much work in progress etc. Simply put, work in progress is not always *actually* in progress. Flow efficiency shows how often that is true. Flow efficiency can be calculated like so:



Teams that are not aware of this metric generally have a flow efficiency of 15%, compared to the acceptable 40+% described by David J. Anderson in ”Low Flow Efficiency: Resist temptation to design out waste”. Once you dive into the details of the work, you can begin to discover the causes of the unnecessary wait times and the impact each one of those causes has. At that point, you can start to design experiments to try to reduce the wait and increase your flow efficiency.

Flow Distribution categorizes work into different work types which supports changing work priorities and report data filtering. It helps to show targeted (and historical) proportion of work item types, brining visibility to planned work allocation.



Commits are also an excellent way of gauging developer productivity as well as the impact of their code. For example, a commit with 100 lines of additions, deletions or changes has little impact to the overall software and probably did not take much time to implement. However, a commit showing multiple changes spanning across multiple files indicates complexity as some planning and thought would likely have to have been done by the developer.

**Computational Platforms Available**

Almost every software-oriented organisation uses GitHub to host their code base whether that’s private repositories for internal use or public open source projects. The platform offers some useful tools to help identify insights about the development lifecycle of a project and some useful analytics of the team behind it. Commits can help draw conclusions about a developer’s behaviour, work style, competence, productivity and more. They show how many lines of code were added/deleted and what changes were made. Valuable team members can also be identified by analysing who is making the most effective pull requests, which team members are doing code reviews to accept those pull requests and which team members are solving open issues in the project.

Gitcolony is an excellent that can integrate with GitHub to provide even more powerful tools

Gitcolony

* Jira
* Testrail

**Algorithmic Approaches**

* Halstead metrics
* AI

**Physical Approaches**

* Team ethics, work style
* Agile
* Scrum

**Ethical Concerns of gathering such data**

**Conclusion**