Measuring Software Engineering

**Introduction**

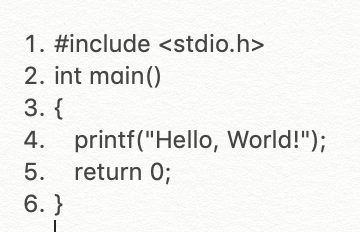
**Measurable data**

* **Lines of code**

One of the most common measurements of how much work is done by one software engineer is the LOC (lines of code) that they write. It’s a simple measurement to measure. Basically the more lines of code that they write the better they are. It sounds like the perfect measurement in theory but in reality it has its flaws. For example it’s not fair to measure one software engineer who codes in Python against a software engineer who codes in C. If they were both write the simple hello world program there is a big difference in the LOC



Printing Hello World in Python



Printing Hello World in C

It takes 6 LOC to do it in C whereas you can do it in one line in Python. This doesn’t mean that the C software engineer is any better than the python one because they are typing more lines of code, this would be an unfair comparison. Although the LOC measurement can be biased towards certain languages it is one of the best measurements from measuring software engineering.

* **Number of bugs per 1000 lines of code**

The industry average of bugs per 1,000 line of code is between 15 and 50 bugs, this figure depends though on when the code is actually typed. The basic human error rate is on average 5%. This figure does depend on many different factors such as the level of testing within the code. This can also be another good level of judging the software engineer. If they are writing code that’s riddled with bugs then there will be precious time wasted debugging the code leading to inefficiencies.

* **Level of testing**
* **Number of commits**

Another way of measuring the software engineering process over time is the number of commits. The number of commits over time should be fairly consistent. It can also be good to gauge how someone works for example do their number of commits in a week deteriorate at the end of the week.

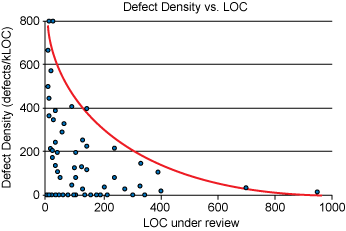
* **Test Coverage / code coverage**

Another method used for measuring software engineering is the code coverage. Code coverage measurement simply determines which statements in a body of code have been executed through a test run. When writing tests for code the ideal code coverage is 100% this means that every line of the code has been tested. From our code coverage we can tell if we have enough testing in place.

https://confluence.atlassian.com/clover/about-code-coverage-71599496.html

* **Code Reviews**

Code review is a software quality assurance activity in which a human will check a program mainly by reading and inspecting the source code. The ideal review is before 400 lines of code have been written since the last code review. There has been a study that showed after 400 LOC the ability to find defect diminishes. In the study it showed that over a 60-90 minute review the yield was on average 70-90% in defect discovery. It is also suggested that you shouldn’t spend more than 60 minutes reviewing code. The optimal time and amount seems to be spending 50minutes reviewing 300 LOC.



https://smartbear.com/learn/code-review/best-practices-for-peer-code-review/

**Computational platforms available**

* **Github**
* **Version Control**
* **Commits and version control**
* **Slack / Teams**
* **Trello**

**Algorithmic approach**

* **Cycle time**

Cycle time is defined as “The total time that elapses from the moment when the work is started on a task until its competition”. The difference between lead time and cycle time is that cycle time is from when the tasked is created through till the work is completed whereas the cycle time is from when the work is started till the time the work is completed. The cycle time data can be used for future projects for gauging how long it’s going to take a certain team to complete a required task. It can also be used to gauge progress in seeing progression of seeing if your cycle time is getting shorted. If your cycle time is shortening over time it’s a sign of good efficiency. Measuring your cycle time is a very straight forward process. All you need to do us record when the task was started and when the task was completed.

* **Waterfall**

It is a fairly straight forward to implement in a new software project. It is a step by step process. There is six stages to waterfall methodology. They are :

1. Requirements

During this initial phase, the requirements of the project are noted and taken down. The document that it gets written on is called a requirements document. It defines what the application should do but not how it should do it.

1. Analysis

During this stage the system gets analysed in order to properly generate the models and business logic that will get used in the application.

1. Design

This stage covers the technical design requirements. In this stage you choose which programming language to use and technologies that you will use in the process. At the end of this stage a design specification report will be generated in which it explains how exactly the logic will be technically implemented.

1. Coding

In this stage the source code is written. This will implement all models and logic that were specified in previous stages

1. Testing

This stage involves QA’s(Quality Assurance) and beta testers to test the source code that’s been written to try and find any bugs within the code. These people report back with any bugs that are found.

1. Operations

In the operations stage the application that has been developed is now ready to be released into a live environment. This stage also involves support and maintenance in order to keep the application up to date and free from bugs.

* **Agile methods**

Agile methods are used to find ways to enhance the software development process, both lead times and cycle times are taken into account. The lead time is the time taken by a team to generate ideas, then develop that idea before delivering the software product, whereas the cycle time is the time from developing the software and deploying it into production.

* **Scrum**

Scrum is a framework within which people can address complex adaptive problems while creating and delivering products to a customer at the highest possible value. Scrum involves a team working together. A scrum is the actual time period when the team works together to finish an increment. The team completes a scrum cycle which can range anywhere from one week to a four week cycle. Within that cycle the team will start off with a sprint planning in which they will set our their targets and the time they want to have it completed by. These goals are set out by the scrum master. By the end of the meeting the full team should be clear on what can be delivered in the sprint and how the increment can be delivered. The team usually will have a daily scrum which will involve a short meeting of 15 minutes in which they reflect on the work that’s been produced in that day. This helps ensure that each member of the team is on the same page on what needs to be done. The team then has a sprint review at the end of the sprint. In this review they review the work that’s been carried out. The last part of the scrum is called sprint retrospective. The goal is that after the sprint cycle the team will either increment(show what’s been done in the scrum) of demo the project they have completed to the client. The team will also talk about what worked and didn’t work in the scrum. This creates a place where the team can focus on what needs to be improved for the next time.

* **Algorithm Analysis**

**Ethics concerns**

**Bibliography**

<https://diceus.com/top-7-software-quality-metrics-matter/>

<https://www.quora.com/What-is-the-average-ratio-of-bugs-to-a-line-of-code>

<https://www.scrum.org/resources/what-is-scrum>

<https://www.atlassian.com/agile/scrum>

<https://www.cgi.com/us/en-us/life-sciences/blog/12-principles-of-agile-methodologies>

<https://airbrake.io/blog/sdlc/waterfall-model>

<https://screenful.com/blog/software-development-metrics-cycle-time>