



Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

SOFTWARE ENGINEERING CSU33012

MEASURING SOFTWARE ENGINEERING REPORT

2 January, 2022

Utkarsh Gupta

Contents

Introduction	2
Software Engineering Process.....	3
1.Measuring Data.....	3
2.Development Analysis Platforms.....	6
2.1 Lines of Code: CLOC	6
2.2 Commits: GitHub and GitHub API	7
2.3 Code coverage: Coverage.Py.....	7
2.4 Test cases passed: Pytest.....	8
3.Algorithmic Approaches.....	9
4.Ethical Analysis.....	10
References:-	11

Introduction

In today's world, the collection of data is becoming increasingly valuable. It is not only useful for a company's operations and products but also its management. Accumulating data and making sense of what it truly represents can have a huge impact on how companies run.

Software Engineering is a relatively new field. There are no guidelines by which software engineers work. This can lead to negative factors such as time wastage, inefficient coding or little to no contribution. Measuring productivity can help a company to evaluate its employees and promote a competitive work environment. Measuring productivity is easy, however, doing it efficiently is difficult.

A company might want to measure productivity of its engineers:

- To develop competitive environment and standards
- To track their progress over time
- To offer insights about how the productivity can be improved

In the end, the aim of measuring software engineering is to provide useful information and enhance the managing of projects. Therefore, measuring SE should be a well-defined and pre-planned process.

Brooks Law states that "Adding manpower to a late software project makes it later". This means that if a project is already late, then, adding more members to its team will only delay it further. So the aim of a team should be to get the process right from the very beginning. Before collecting any data, the company must determine its goal and the methods of measurement must be carefully defined. Moreover, it is important to collect this data ethically for the well-being of the company and its employees.

With this in mind, this report tries to build a broad view of software measurement and how one might use it to evaluate employee productivity.

Software Engineering Process

1.Measuring Data

- [Source line of code](#)

One way to measure software engineering is to count each physical line in the source code. While this is not indicative of the quality of the code nor its stability, it can signal that the programmer is working regularly on the code.

This metric can be measured in two ways. The easiest way is to measure the PLOC (Physical Lines of Code) i.e. the total number of lines written. Let us see an example.

```
for (int x =0;x<=4;x++){  
    a=b+c;  
    print(a);  
}
```

As we can see above there are four lines of code. The loop can occur however many times, but the PLOC will remain the same. This method is however quite easy to manipulate. By just adding print statement and brackets, the programmer can increase their lines of code. Therefore formatting a code has big impact on this method.

The other method to measure the lines of code is a bit more complex. This method is called LLOC (Logical Lines of Code). In simple terms, this method aims to count the number of statements that are executed. This is not a very good metric as it can vary from language to language and does not have a very robust definition. In the above code, there are 3 LLOC. The for statement, the addition statement and the print statement.

These methods can be more effective at predicting the productivity if we introduce constraints to them. For example instead of just counting lines of code we can count them each month. So we can have Lines of Code per Month. That may give a better indication of whether someone is regularly producing code.

- Number of Commits

Commit is a change that a programmer makes to a code. Number of commits as a metrics can be useful to see which programmer works most on the code.

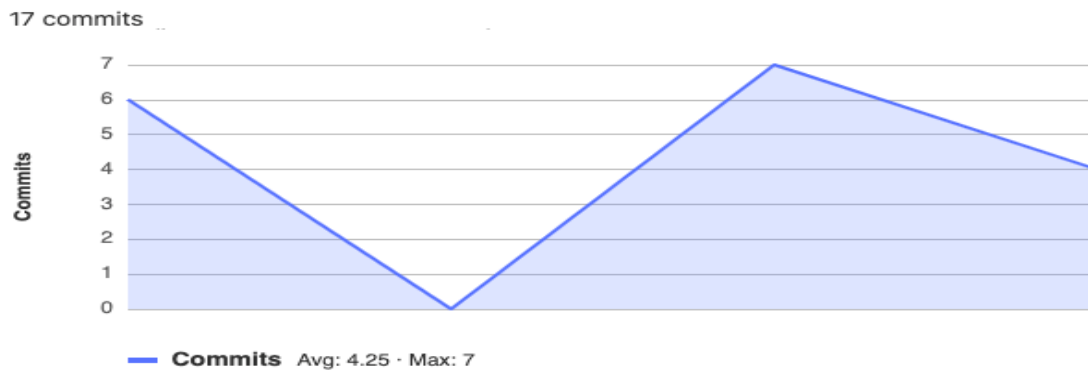


Figure 1: Commits graph

A high number of commits over an interval of period indicates that a programmer is working regularly on the code. A decrease in commits overtime can indicate fall in productivity. Comparing commits between team members of a project can also indicate which member contributes most to the project.

A commit however can be just a statement added to a piece of code or formatting the code. Therefore a commit may not accurately signal that an important new code was added to the overall code. Different programmers have different working styles, they may like to commit their code often or they may write a chunk of code properly first and then commit it.

- Code Coverage

Code coverage allows a developer to understand how much of their code was tested. A high code coverage indicates that the code is less prone to bugs and failures because it has been tested for most of the ways it can execute. A low code coverage on a test suggests that more tests are needed to check the robustness of the program because some code has not been tested and can fail.

Code coverage can use different criteria to see how a code is executed. Some of these are:

1. Function coverage: the number of functions that are executed.
2. Statement coverage: the number of statements that have been executed.
3. Branches coverage: the number of the branches (possible ways a program can execute) of the control structures have been executed. For example if else statements.

For a program to have a good coverage, it should have a high score on all of these criteria. Evaluating productivity based on this method is one of the most reliable ways because coverage can distinguish between developers who develop trustworthy code and those who do not.

- Test cases passed

Testing is a very important part of development. It assures that a code is fail safe and bug safe. Test cases passed is a metric that gives the percentage of the tests that are passed during its testing.

$$\text{Test cases passed} = \frac{\text{No. of cases passed}}{\text{No. of Cases executed}} \times 100$$

Let us say we decide to test our code with 60 tests. Out of those only 30 pass. Then the percentage of code is $30/60 \times 100 = 50\%$.

Here, it is to be noted that the quality of the tests matter. Testing a same piece of code or same conditions can give false passes, therefore it is up to the programmer to assure that their test are written keeping all the possible situation a code can execute in mind.

If the pass percentage remains the same (not including 100%) at latter stage of a project, then we can assume that there are bugs that the programmers have

not been able to resolve. If this percentage reduces at latter stages is a signal that there are bugs in the code previously written and should be looked into.

2. Development Analysis Platforms

There are various tools available to developers that can help them to measure these metrics. They differ from language to language, but provide the same metrics.

2.1 Lines of Code: CLOC

CLOC(Count Lines Of Code) is a package that can count the Physical Lines Of Code of various languages including C++,Java, Python and more. It is written in Perl with no additional dependencies and therefore is highly portable. To install CLOC with the help of npm one can do:

```
npm install -g cloc
```

The advantage of using CLOC is that results from many runs can be summed by the language and the project. Below is a sample output from CLOC:

Language	files	blank	comment	code
C++	615	93609	110909	521041
XML	27	564	23	4107
SQL	18	517	209	3433
Assembly	12	161	0	1304
Javascript	3	70	140	427
HTML	1	7	0	250
SUM:	676	94928	111281	530562

Figure 2. An output by CLOC showing coverage of code

2.2 Commits: GitHub and GitHub API

GitHub is a platform that hosts code for version control. It keeps track of the commits that developers make and makes it easy to evaluate which developers are committing most to the code.

Each user has a graph on their profile that shows the frequency of their commits in the past year.

GitHub also provides statistics for its various repositories through GitHub API. With GitHub API we can extract information like commits, contributions, code additions etc. We can use this information using tools like PyGitHub and compare them using libraries like matplotlib and pyplot. Below is an example of the commits plot similar to GitHub but made by gathering data through its API using PyGitHub and Plotly.

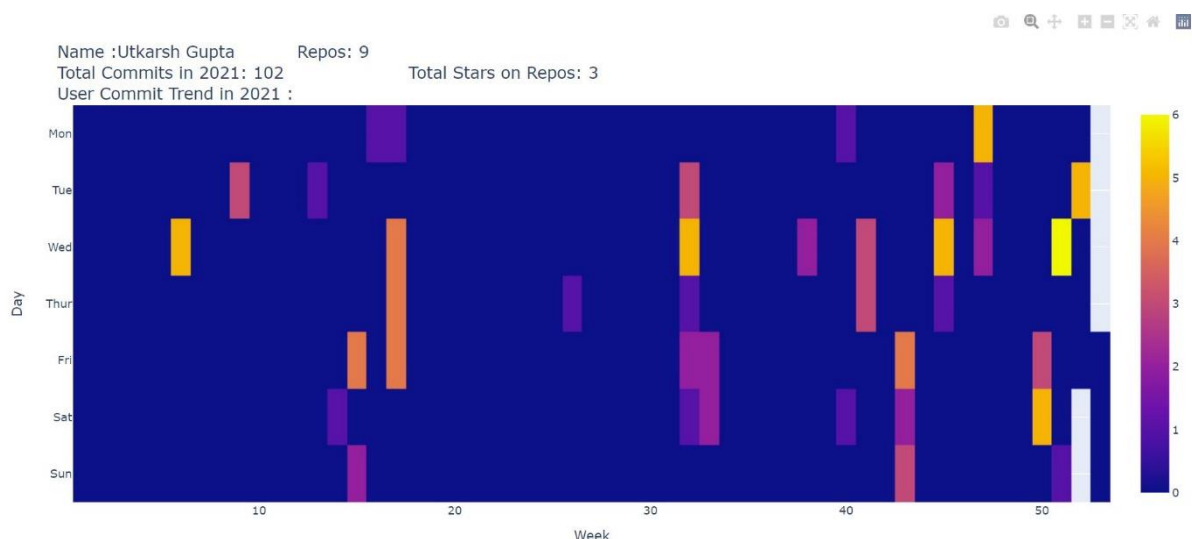


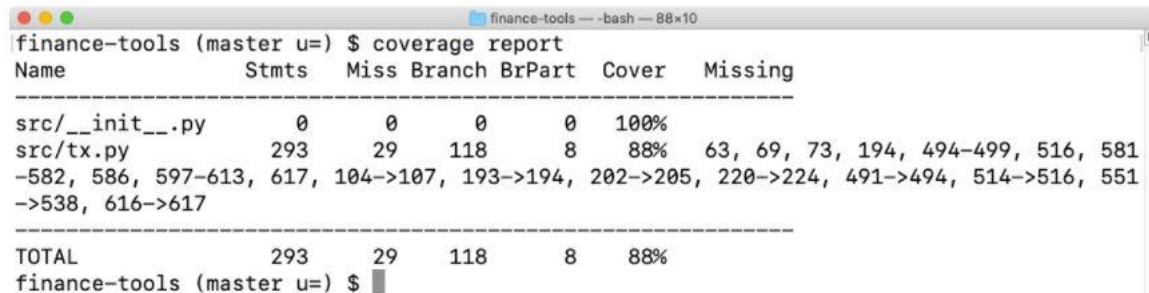
Figure 3. A commit graph of user made using GitHub API and Plotly

Using this tool, one can get a clear indication of how productive a developer has been over the past year.

2.3 Code coverage: Coverage.Py

Coverage.py is a tool to measure the coverage of code in Python language. For other languages similar packages are available. As mentioned earlier coverage can be of many criteria. Coverage.py's default capabilities allow the user to

measure statement coverage. Additionally, it can also measure branch coverage and can tell which lines were executed for which tests.



```

finance-tools (master u=) $ coverage report
Name                               Stmts   Miss Branch BrPart  Cover   Missing
-----
src/__init__.py                     0       0      0      0    100%
src/tx.py                          293     29    118      8     88%  63, 69, 73, 194, 494-499, 516, 581
-582, 586, 597-613, 617, 104->107, 193->194, 202->205, 220->224, 491->494, 514->516, 551
->538, 616->617
TOTAL                              293     29    118      8     88%
finance-tools (master u=) $

```

Figure 4. Execution of Coverage.py

Figure 4 shows the output of coverage's execution. As we can see it informs the user of the branches and statements covered. In this case, only 88% of the code was covered in the file src/tx.py.

2.4 Test cases passed: Pytest

Different language have different tools for unit tests. In this report our focus will be on pytest, a tool to test code in Python language.

Pytest is a testing tool for python. It makes it easy to execute small tests and allows for complex functional testing of applications.

Lets see an example:

```

# test_sample.py

def inc(x):
    return x + 1

def test_inc():
    assert inc(3) == 5

```

Here, we are executing a function to subtract one from the input and then comparing it against the correct result using the assert statement. In pytest this code can be run using the command 'pytest'.

The sample output for this looks like this:

```
===== FAILURES =====
test_answer

def test_answer():
>     assert inc(3) == 5
E       assert 4 == 5
E       + where 4 = inc(3)

test_sample.py:6: AssertionError
===== short test summary info =====
FAILED test_sample.py::test_answer - assert 4 == 5
===== 1 failed in 0.12s =====
```

Figure 5. An output of pytest on test_sample.py

As we can see in Figure 5, the test fails because of a different answer than expected. It also tells us what went wrong, in this case assert statement failed.

3. Algorithmic Approaches

Machine learning is a field that has grown a lot in the past few years. In simple terms, machine learning allows the computers to analyse data without explicit programming. It can be done in two ways: supervised and unsupervised learning. In supervised learning a model is trained by feeding it training data that contains inputs and outputs. In unsupervised learning, no training data is given and the data is analysed solely based on the inputs. The model itself has to learn how it wants to handle the data.

Measuring productivity can be achieved very well with supervised learning. Let us say we have collected metrics of some employees in a company. Now we can manually label some of them as good or bad. We then run these inputs and outputs in a supervised machine learning algorithm that will be trained in this data. Now the algorithm will be able to decide if the new inputs that are given are good metrics or bad without needing any more outputs. An example of this is K Nearest Neighbour algorithm that clusters the datapoints based on the similarity of their variables.

For unsupervised learning we can use clustering methods, such as K Means or hierarchical clustering. These techniques make clusters of data according to

their similarity without needing any training data. Therefore we will be able to distinguish between the good and the bad metrics.

One of the best tools for machine learning is Google's TensorFlow. It is an open source machine learning platform that makes it easy to build and deploy machine learning powered applications. We can label our data as outputs and inputs and specify the number of "neurons" we want to have in our learning algorithm. Neurons take some input, apply a function to that and give an output. TensorFlow allows for scalability and therefore is a very good tool.

For clustering, we can use platforms such as RStudio which make it easy to call functions for K nearest neighbour and hierarchical clustering.

4. Ethical Analysis

Working with big data that contains information about employees, brings the question about its ethicality in sight. We need to understand that the data might contain personal information that employees do not want disclosed. Further keeping this data fully secure is the responsibility of the company. According to Harvard Business Review "only 30% of the executives whose companies use workforce data reported being highly confident they are using the data responsibly".

Collecting data such as Lines of Code, Number of Commits, Code coverage and Tests Passed do not fall into the category of personal data as they do not tell anything significant about the employee.

However, when saving this data to a database it is good practice to encrypt the names of the employees. Tools such as Faker in Python can help to keep names of employees safe by storing fake names into the database. This can also be used to hide other categories such as age and gender.

No algorithmic approach is best or even correct to measure the ability of a programmer to code. For machine learning approach, we do not know what is behind the black box that tells us whether a programmer has been productive or not. Moreover many of these metrics are very easy to manipulate and as discussed might not give a true image of someone's ability.

Therefore, relying heavily on data to measure someone's productivity is not a very good approach. Continuously collecting data about someone's work might not be best suited to all employees as people like to get a sense of freedom and privacy when they are working. Some people are even more productive when they are not being evaluated every second. Monitoring people can elevate stress and lower workforce satisfaction which signals that gathering metrics can have a negative effect on productivity. To sum up, evaluating people from collection of data should not be the goal of the company, rather improving its employees' performance.

References:-

1. <https://www.atlassian.com/continuous-delivery/software-testing/code-coverage#:~:text=Code%20coverage%20is%20a%20metric,get%20started%20with%20your%20projects.>
2. <https://reqtest.com/agile-blog/agile-testing-metrics/>
3. <https://github.com/AlDanial/cloc>
4. <https://zapier.com/engineering/github-streak-habit/>
5. <https://stackify.com/code-coverage-tools/>
6. <https://coverage.readthedocs.io/en/6.2/>
7. <https://docs.pytest.org/en/6.2.x/>
8. <https://mincong.io/2019/09/02/measure-coverage-with-coverage.py/>
9. <https://www.geeksforgeeks.org/machine-learning/>
10. <https://hbr.org/2019/02/how-companies-can-use-employee-data-responsibly>