Measuring Software Engineering

CS3012: Software Engineering

Table of Contents

1 Introduction

In this report I will discuss the ways in which we can measure the software engineering process and the benefits of these measurements. I will also discuss some of the available algorithmic approaches to carrying out these measurements and the available platforms that facilitate these processes. Finally, I will briefly outline some of the ethical concerns surrounding this field.

1.1 What Is Software Engineering?

Merriam-Webster succinctly defines *software engineering* as: *"a branch of computer science that deals with the design, implementation, and maintenance of complex computer programs"*. Using this definition we can see that the field of software engineering is of key importance in the 21st century as some of the largest companies in the world depend primarily on *"complex computer programs"* to turn a profit, e.g., Apple, Amazon, Google and Samsung.

Given that the software development process is so heavily relied on by some of the largest companies in the world and that the primary goal of these companies is to maximise profits it should be of no surprise to anyone that a large number of resources have been put into making the process of developing high-quality and well-tested software as efficient and effective as possible.

1.2 Why Do We Measure It?

In order to achieve efficient and effective development the software engineering process must be measured, tracked and analysed. This could allow development teams to identify areas of low productivity, improve the accuracy of time and cost estimates and to prioritise the solving of issues more effectively.

The measurement of software engineering involves the use of software metrics. A software metric is some characteristic of the software that is quantifiable or countable, e.g. the number of lines of code written or the amount of issues raised and resolved.

Through the use of these metrics teams can measure their productivity and performance, identify bottlenecks in the development process and accurately estimate the time-to-completion of projects. Team leaders can use these metrics to improve the productivity of teams through the identification of issues in the development process.

They also allow team leaders to more accurately estimate development time and costs and to identify areas that could be improved upon[1].

2 Choosing Software Metrics

In order to measure the software development process we must first decide on the appropriate software metrics to use. In this section I will outline a number of commonly used software metrics as well as their respective benefits and drawbacks.

2.1 Code Churn

Code churn measures the number of lines of code modified, added or deleted between software releases or over a set period of time[2].

Benefits:

- ➢ Code churn is very simple to measure and visualise using most popular version control systems.
- ➢ It can be tracked individually for each developer working on a project.
- ➢ Allows long-term trends in the volume of code being produced to be easily identified.

Drawbacks:

- ➢ The volume of code written does not correlate directly with the volume of effective or high-quality code being produced, i.e. ten good lines of code might be better than one hundred poor lines of code.
- ➢ Not all lines in a file are actually code (whitespace, comments, etc.).
- ➢ Solely using code churn as a software metric might encourage developers to add redundant or unnecessary code in order to artificially increase the volume of code they produce.

2.2 Unit Tests and Code Coverage

The number of unit tests created during a development task as well as the accompanying code coverage of those tests can be measured.

Benefits:

- ➢ Unit tests ensure that code produced is free from bugs and of a high quality.
- ➢ Ensuring a very high code coverage also prevents bugs and can be used to remove/simplify less useful lines of code.
- ➢ Can be combined with code churn (§2.1) to reduce the volume of unnecessary code being produced leading to more efficient development.

Drawbacks:

- ➢ There could be a significant overlap in the code being tested by different unit tests leading to wasted development time and redundant lines of test code being produced.
- ➢ Code that successfully passes tests may still be entirely unnecessary in the broader context of the software.

2.3 Issues Raised/Closed

The number of issues/bugs raised in a specific period of time combined with the ratio of issues resolved to issues raised can be measured.

Benefits:

- ➢ Prioritises the production of bug-free code especially when combined with the use of unit tests (§2.2).
- ➢ Ensures that issues will be resolved as quickly and efficiently as possible in the development process.

Drawbacks:

- ➢ A high ratio of issues resolved to issues raised may not always be wholly positive as it ignores the fact that a large number of issues are occurring - hence our combination of the ratio with the raw number of issues being raised.

2.4 Conclusion

All of the metrics I described above could be exploited by developers to produce misleading data: redundant lines of code, redundant unit tests, etc. As such, it

would appear that some combination of software metrics must be tracked in order to acquire a reliable measurement of the software development process.

3 Algorithmic Approaches Available

In this section I will briefly discuss two of the available algorithmic approaches to measuring software engineering: a machine learning approach, and functional point analysis.

3.1 Using Machine Learning

Hélie, et al. of Semmle Inc. used a machine learning approach to measure software development productivity[3]. Their aim was to measure the *'coding productivity'* of developers by analysing their commited changes to version control repositories while taking into account the quantity *and* quality of the code produced. They used a dataset of over 10 million commits submitted by ~300,000 developers to ~56,000 open source projects and the *QL* programming language[4] to analyse the quality of committed code.

They attempt to measure the quantity of code produced not through the raw volume of code committed but through the amount of *'labour time'* taken to produce the code. To do this they train a neural network to predict, for each minute a particular developer spent between two seperate code commits (a *'commit interval'*), what is the probability that the developer was actually coding during that time. They train the neural network using recent data for that developer from the previously mentioned dataset.

Combining both of these models allows them to garner in-depth quantitative analysis of the software development process.

3.2 Functional Point Analysis

Functional Point Analysis (FPA) was developed by Allan J. Albrecht in 1979 at IBM. The initial definition given by Albrecht for FPA was: *"... a dimensionless number defined in function points which we have found to be an effective relative measure of function value delivered to our customer"*[5]. FPA is used to measure the

'functional size' of software, i.e. the amount of functions it provides to its users. Tracking this quantity over numerous releases allows one to measure the overall progress of the software development.

FPA can be used to measure the ratio of functionality provided to the functionality requested by the user or customer. It can also be used to accurately estimate the time and cost required to implement a specific number of functions using previously acquired data.

4 Overview of Available Platforms

4.1 GitPrime

GitPrime[6] is a commercial tool that can be integrated with git that provides visual insights and reports for individual developers as well as for entire projects. GitPrime can be used by teams to track commits, pull requests, issues and more. It aggregates repository data into numerous dashboards which are intended to make the management and tracking of a project's progress much clearer.

GitPrime also calculates relevant statistics for a project such as how responsive or receptive issue submitters are to comments, what activities developers have been focusing on over a specific time period and the average time-to-resolve of pull requests.

4.1 Clockify

Clockify[7] is a free time tracking tool that development teams can use to track the time spent by developers on different tasks and projects. Like GitPrime, Clockify provides dashboards which contain data visualisations that make the tracking of progress much clearer.

Clockify also allows provides users with the ability to track a project's profits and costs via detailed reports. It allows teams to set project estimates and goals.

5 Ethical Concerns

Some ethical concerns arise when discussing how the measurement of software engineering could be implemented.

5.1 Incorrect Use of Software Metrics

If a narrow selection of software metrics are used to measure the development process some developers may be unjustly marked as unproductive due to the ineffective constraints being put upon them by the software metrics. For example, if one developer writes a moderate amount of high-quality code and another developer writes a large amount of mediocre code then a team that uses only the *lines-of-code* metric will incorrectly mark the first developer as less productive than the second developer. These kinds of situations are ethically dubious as they unfairly single out developers and punish them.

Therefore, it is very important that the combination of software metrics being measured as well as the platforms/algorithms being used to measure them are selected fairly and correctly.

5.2 Data Privacy

Another ethical concern related to the measurement of software engineering is that of data privacy. The types of data that are being collected and analysed by teams may, in some cases, infringe upon a developer's privacy. It is important that managers are completely transparent when it comes to this process and make it clear to the developers what kind of metrics are being used and what data is being tracked.

6 Closing Remarks

In this report I have discussed why it is quite important for development teams to measure the software engineering process. I have also outlined a number of ways in which this measurement can be carried out as well some of the platforms available for doing this. Finally, I outlined some of the ethical concerns related to this field.

In my opinion, the use of these measurement tools and techniques by development teams is one of the most important aspects of the software engineering process and I hope to have made this quite clear in my report.

References

[1] https://stackify.com/track-software-metrics/

[2] https://codescene.io/docs/guides/technical/code-churn.html

[3] https://semmle.com/assets/papers/measuring-software-development.pdf

[4] http://drops.dagstuhl.de/opus/volltexte/2016/6096/

[5] https://www.geeksforgeeks.org/software-engineering-functional-point-fp-analysis/

[6] https://www.gitprime.com/

[7] https://clockify.me/