MEASURING SOFTWARE ENGINEERS

CSU33012 Software Engineering

Abstract

To deliver a report that considers the ways in which the software engineering process can be measured and assessed in terms of measurable data, an overview of the computational platforms available to perform this work, the algorithmic approaches available, and the ethics concerns surrounding this kind of analytics.

Mary Coyne

coynem2@tcd.ie

Introduction

Measurement helps managers spot scenarios when “things do not add up” due to unclear or conflicting project requirements. A study conducted back in 2008, estimated that 68% of surveyed companies were statistically unlikely to have a successful project due to poor project requirements. More specifically, they were likely to face the following issues:

- Have a budget of 160% more than the original.
- End up with 180% more of estimated time than expected.
- Deliver less than 70% of the expected functionality.

(Infopulse, 2019)

In this essay we are going to focus on the ways in which the software engineering process can be measured and assessed in terms of measurable data, an overview of the computational platforms available to perform this work, the algorithmic approaches available, and the ethical concerns surrounding this kind of analytics and how these stop the figures above from happening.

How to measure software engineering and what you are measuring

Direct metrics are important because it is presumed valid and other metrics are validated in terms of it. Some common derived metrics in software engineering are:

- Programmer productivity (code size/ programming time)
- Module defect density (bugs / module size)

- Requirements stability (number of initial requirements /total number of requirements)
- System spoilage (effort spent fixing faults / total project effort)

To measure productivity of software engineers', managers can use two types of metrics. The first is size-related metrics indicating the size of outcomes from an activity. For instance, the lines of written source code. The second is function-related metrics represent the amount of useful functionality shipped during a set period of time. Function points and application points are the most commonly used metrics for waterfall software development, while story points are the usual metrics for agile projects. The productivity metrics chosen should be consistent, auditable, available and repeatable.

Defect density is the number of defects confirmed in software/module during a specific period of operation or development divided by the size of the software/module. It enables one to decide if a piece of software is ready to be released. Defect density is counted per thousand lines of code also known as KLOC. Stability Testing is the testing process to determine the ability of the software product to perform its required functions under specified conditions for a stated period of time, or for a certain number of operations. In other words, it is a testing of the stability of a software product.

Metrics will be used to optimise software delivery, measure software maintainability and measure the requirement effectiveness.

Optimise software delivery

Sprint burndown is a key metric. A burndown report shows the complexity of the work done throughout sprints. By tracking this metric, you can obtain important insights such as if

there is a lack of scheduled work for one sprint or a gap in planning. Team Velocity metric accounts for the amount of software the team completes during a sprint. By measuring velocity you can set better delivery expectations and realistic sprint forecasts, understand if your team is blocked, spot unforeseen challenges that were not accounted for during sprint planning, and investigate if your process changes have any results. Throughput indicates the total value-added work output by the team. Measuring throughput also helps to detect when the team is blocked as the throughput metric drops and understand when the team is overloaded if you compare the average throughput against the current workload. Cycle Time stands for the total time that elapses from the moment when the work is started on an item until it is completed. This metric allows you to estimate how fast you can deliver new features to users. It's also another way to understand the team's speed for different tasks by breaking the total throughput down to median time by status or issue type.

Measure software maintainability

Lead time is the time between the definition of a new feature and its availability to the user. It helps you estimate how well your team is performing so far. Mean time to repair (MTTR) shows how fast can you deploy fixes to the consumers. Code coverage shows the amount of code measured in LOC that is covered by a unit test. Bug rates is average number of bugs that are generated as new features are being deployed. It can help you estimate whether you are delivering value or just deploying some half-baked code for the sake of frequent deployments.

Measure the requirement effectiveness

Using these metrics, the goal here is to make sure that the team can work at the consistent pace when presented with both static and dynamic requirements. Task volume + average estimates is the number of tasks your team can complete in the face of change, compared against the average estimates will help you understand how consistently your team is completing their work. Recidivism is a good indicator of incomplete or inconsistent requirements that you may want to investigate. A high number means someone in the workflow didn't have the same standard as someone else downstream.

Platforms used

There are many platforms used to measure software engineering. When researching platforms that are used to capture the metrics of software engineering there are many articles titles "top 20 best software engineering tools 2020" and other similar titles. According to these articles there are many platforms that can be used to process these metrics, and a lot of software people use for gathering it and processing it.

Development tools can be of many forms like linkers, compilers, code editors, GUI designer, assemblers, debugger, performance analysis tools etc. There are certain factors to be considered while selecting the corresponding development tool, based on the type of the project including company standards, tool usefulness, tool integration with another tool, selecting an appropriate environment and learning curve. Whatever tool you select will effect the project's success and efficiency.

GitHub, monday.com, Embold, Linx, Quixy, Atom, Cloud 9, NetBeans, Bootstrap, Node.js, Bitbucket, CodeCharge Studio, CodeLobster, Codeenvy, AngularJS, Eclipse, Dreamweaver,

Crimson Editor, Zend Studio, Jira, CloudForge, Azure, Spiralogics Application Architecture (SAA), Delphi, and Zoho Creator (Software Testing Help, 2020) are all popular software development tools that are widely used throughout the industry. Some of these tools fix bugs before deployment which saves a lot of time and energy in the long run. The tools accelerate the design, development and automation of custom business processes, including easy integration of applications, systems and databases. They will let you integrate your existing software development tools and your data from multiple tools. You can automate the workflows and save your time.

GitHub, for example, is a powerful collaboration tool and development platform for code review and code management. With this GitHub, the users can build applications and software, manage the projects, host the code, review the code etc. Developers can easily document their code and can host the same from the repositories. Few features of GitHub that make it a useful tool are its code security, access control among the team members, integration with other tools and there are many others. It can be hosted on servers and on a cloud platform.

The algorithmic approach

It is possible to design software metrics in an empirical manner, as it has been done from the beginning of their development. Another way is to create a theoretical approach and construct software metrics in a systematic fashion. The most popular software metrics, which are discussed heavily today, and which were created in the middle of the seventies are the measures of McCabe and a cluster of measures of Halstead. All computer programs

are algorithms. So, to find complexity of a computer program, it is natural to use algorithmic measures of complexity.

The complexity measure that was proposed based on the control flow graph (CFG) model of a program P is called the cyclomatic number V(P) of a program P and is defined by the formula: V(P) = e – n + 2, where e is the number of edges and n is the number of nodes in the CFG of a program P. The cyclomatic number v(G) is equal to the number of independent cycles in G and it is also a function of the number of predicates in the program.

Direct complexity measures usually measure processes and programs that are completed. However, it is often necessary to make these kinds of estimates before the program or process is completed. In this case, a complexity of a problem is needed. The complexity of a problem often has differences from the complexity of its solution. Complex solutions may exist for simple problems. Many important problems that have hard solutions have low problem complexity ("their Kolmogorov complexity or algorithmic information is rather low"). (Debnath, N; Burgin, M. N/A)

"All software (resource) metrics are algorithmic complexity measures and any complexity measure represents some resource software metrics" (Debnath, N; Burgin, M. N/A)

Ethics

Ethics defines a system of moral principles that people live by. While being ethical is the right thing to do, a lot of people and companies are unethical in their practices. Software engineers are obliged to be ethical in all of their work because all human beings are obliged

to be ethical in their work and practices and to all other human beings throughout their lifetime.

Software engineers must concern themselves primarily with the health, safety and welfare of those who are affected by their work. But we need to broaden our understanding of a number of aspects of this claim, including the types of harms the public can suffer as result of this work, how software engineers contribute to the good life for others, who exactly are the 'public' to whom the engineer is obligated, why the software engineer is obligated to protect the public, what other ethical obligations software engineers are under, how software engineers can actually live up to ethical standards, what is the end goal of an ethical life in software engineering, and what are the professional codes of software engineering ethics.

Failures of critical software systems can result in catastrophic loss of life or injury to the public. If such failures happen, directly or indirectly, because of a software engineers' choices to ignore their professional obligations, then these catastrophes are unfortunately the consequences of unethical professional behaviour. It was unethical because the software engineer could have prevented these things from happening but did not. They may have been lazy, did something the 'easy' way, ignored some problems or something else. Whatever it was, was not how the job should have been handled.

Ethics is not just about avoiding harms and mistakes, as a narrow focus on preventing catastrophic events might make us believe. Ethics is also about doing good and making the world a better place. Software engineers bring positive contributions to our world all the time. A lot of the work they do makes everyday life easier and more enjoyable for

everybody else, especially with all the advancements in the tech sector contributing to everyday life.

Stakeholders are the people who are affected by the product that is being worked on. This includes the software engineer, their boss, whoever will be using the end product, etc. The interests of all stakeholders may not always align which can cause problems. For example, the employer's interests in cost-cutting and an on-time product delivery schedule may frequently be in tension with the interest of other stakeholders in having the highest quality and most reliable product.

Conclusion

There are many areas in which a software engineer can be assessed and measured. It is necessary to assess software engineers in these many different areas such as measurable data, computational platforms, algorithmic approaches and their ethics. This in turn makes sure that the work is done to the best of the software engineers' ability and is done in a productive and safe way which will lead to the best outcomes for everybody with the software or project that the software engineer was working on.

References

Caner, C.; Bond, W. P. (2004) *Software Engineering Metrics: What Do They Measure and How Do We Know?* http://www.kaner.com/pdfs/metrics2004.pdf

Debnath, N; Burgin, M. (N/A) *SOFTWARE METRICS FROM THE ALGORITHMIC PERSPECTIVE*

http://cs.ndsu.edu/~perrizo/saturday/papers/paper/CATA-2003%20Papers/206.pdf

Infopulse (2019) *Top 10 Software Development Metrics to Measure Productivity*

https://medium.com/@infopulseglobal_9037/top-10-software-development-metrics-to-measure-productivity-bcc9051c4615

Software Testing Help (2020) *20 BEST Software Development Tools (2020 Rankings)*

https://www.softwaretestinghelp.com/software-development-tools/

Vallor, S. (N/A) *An Introduction to Software Engineering Ethics*

https://www.scu.edu/media/ethics-center/technology-ethics/Students.pdf