CS3012 Software Engineering

**Mary McDonald 15316382**

# Measuring Engineering - A Report

## Introduction

In this report I will discuss the different ways in which the software engineering process can be measured and assessed. After I examine what the 'Software Engineering Process' itself is, I will review the measurable data and all its complexities. I will then go over a number of the computational platforms available to perform this analysis and some of the algorithmic approaches available. Lastly I will write about ethics concerns surrounding this kind of analytics - mostly to do with privacy and security of both the client and the engineer.

## What is the Software Engineering Process?

A software process (also known as software methodology) is a set of related activities that lead to the production of the software. (These activities may involve the development of the software from the very start or modifying an existing system.) Any software process must include the following four activities:

1. **Software specification** (or requirements engineering): A Software specification is usually a detailed document detailing things such as the purpose of the Project, the Scope and Overall Description, the Operating Environment, any Assumptions and Dependencies developers may have, Design and Implementation Constraints, System Features, External Interface Requirements and any other Requirements.
2. **Software design and implementation**: Software design and implementation is the stage in the software engineering process at which an executable software system is developed. Software design is a creative activity in which you identify software components and their relationships, based on a customer's requirements. Implementation is the process of realizing the design as a program.
3. **Software verification and validation**: This is essentially software testing, to test whether a software system meets the specifications and that it fulfills its intended purpose or not. It may also be referred to as software quality control- the software must conforms to it's specification and meets the customer needs.
4. **Software evolution** (software maintenance): The modification of a software product after delivery to correct faults, to improve performance or other attributes. A common perception of maintenance is that it merely involves fixing defects, however this is

not the case. Software maintenance can be categorized maintenance activities into four classes:

- Adaptive – modifying the system to cope with changes in the software environment.
- Perfective – implementing new or changed user requirements which concern functional enhancements to the software.
- Corrective – diagnosing and fixing errors, possibly ones found by users.
- Preventive – increasing software maintainability or reliability to prevent problems in the future.

In practice, these four activities of the Software Process include sub-activities such as requirements validation, architectural design, unit testing, ...etc. There are also supporting activities such as configuration and change management, quality assurance, project management, user experience. Along with other activities aim to improve the above activities by introducing new techniques, tools, following the best practice, process standardization (so the diversity of software processes is reduced), etc.

From my research I have learned that the Software process is complex, and also involves a lot of human thought. There is no ideal process and it is extremely common for companies to develop their own Software Processes.

**Measurement and Assessment**

Measuring developer performance is an inexact science with a lot of controversy. Anyone measuring Engineering process can face more pushback from developers as they may feel their managers don't have enough knowledge of their work. This enforces the need for intelligible methods of measurement that need to be implemented and clearly communicated. If the whole team is on the same page then the process will be much more easily measured and assessed.

**Measurable data**

*"Any kind of quantitative metrics for software developers tend to actually reduce overall productivity. They also have negative impact on motivation, and will eventually drive good people out."*—vartec

Software engineering could technically be measured in many ways..

- ➢ Such as speed of developer - however any developer can write broken or incorrect code quickly
- ➢ Or lines of code- however longer code isn't necessarily more efficient or better code and some languages require more lines of code then others (e.g. Java vs. Ruby—Java developers are going to have ~10x more lines)
- ➢ Feedback from customers, number of features, ability to ship code, QA, unit tests, memory usage, CPU utilization and ability to meet deadlines, no.of commits

It seems to me the solution is to Measure what matters while it matters. This means that measurement needs to be focused on the right goals for the right time, and you should expect that what you measure will change frequently as timeline of a specific project changes.

Also frequently releasing If you want to report something related to engineering productivity to an executive team, you could start by reporting on release frequency - especially if your frequency is scheduled at once a week or less. The act of releasing frequently forces reckonings throughout your software and process stack. Frequent releases also spot problems for example: If the code is too brittle to release easily, if there are too many places that require manual QA, f you have a complex series of steps required to get code into production, if developers don't break their changes down into clear, well explained chunks.

**Computational platforms available to perform this work**

Although it seems that measuring software engineering can be a complex task, we have not even begun to look at what modern computational platforms are available to do this work today.

Computational platforms and environments for software engineering measurement are platforms of integrated software that have user-friendly interfaces and that can be used by managers/bosses to properly analyse the software engineering process. I will take a look at some said platforms below.

The Personal Software Process (PSP)

The PSP is a structured software development process that is intended to help software engineers better understand their performance by tracking their predicted and actual development of their code. It was created by Watts Humphrey.

The aims of the Personal Software Process are to help engineers:

1. Improve their planning skills.
2. Make commitments they are capable of fulfilling.
3. Manage the quality of their work.
4. Reduce the amount of defects in their projects.

The original version of the PSP was very tedious and required manual calculations which demanded substantial effort (Developers had to manually fill out multiple forms including a project plan, a time recording log and a design checklist.) While this method had various advantages and disadvantages, the manual aspect of the PSP raised concerns about the reliability of the analytics. In response to this concern, the Leap Toolkit was developed.

### The Leap Toolkit

The Leap Toolkit allowed developers to control their own data files in an attempt to avoid flawed metrics as well as maintaining data only relevant to the individual developer’s activities. It also rectified some of the problems with the quality of data being collected by the original version of PSP by automating and normalising data analysis -although it still required manual input. The Leap Toolkit also created a repository of personal process data which enabled data portability.

### Hackystat

We can see from the above computational platforms that perhaps a fully automated approach would be more efficient. The Hackystat framework was created in order to fully automate the data collection process for developers. It is used for collection, analysis, visualisation, interpretation, annotation and dissemination for the software engineering process. The framework has little to no overhead for developers, as sensors have to be implemented into development tools in order to gain insight. Integrating this project into developer's work days included creating editors, build tools and testing tools for client side data collection. Events such as switching files, editing methods, constructing test cases, and invoking tests were all monitored by this software.

## The algorithmic approaches available

### Artificial Intelligence

AI is a term for simulated intelligence in machines. AI methods may be used in software engineering to optimise either the engineering process itself or the products that it produces. Some AI techniques used in the software engineering community today would be using computational search and optimisation techniques to optimize some parts of the software engineering process and classification, earning & prediction as during the early stages of the software engineering process, AI techniques have proved to be particularly useful for modelling and predicting software costs.

We can see from this that AI integration with the software engineering process shows promising results. However, there are still many improvements to be made and the level of usefulness AI will be to the software engineering process depends on how much it is developed in the coming years. As research is carried out all over the world, every day - AI is quickly evolving, and so is the software engineering process and almost every aspect of software development. This is why s many aspects of software engineering has started integrating many of the algorithms, methods and techniques emerging from the evolution of AI.

## Computational Intelligence

The expression computational intelligence (CI) usually refers to the ability of a computer to learn a specific task from data or experimental observation. As we have discussed above, there is evident growth that topics such as AI having an impact on the software development process.

Computational Intelligence is another relatively new area in computational thinking and, much like AI, has also been seen to provide optimisation techniques to the software development process. CI is quite similar to AI with the main difference between the two being that AI is based on hard computing techniques (using a modern chipset) and CI uses a combination of techniques (both hard and soft) including fuzzy logic.

Some CI techniques that could be used in the software engineering community today would be Fuzzy Logic - A branch of mathematics that allows computers to model real world problems similar to the way that people do through the use of fuzzy sets. The degree to which an object is a member of a traditional set is strictly 0 or 1 but with fuzzy sets, can be any value between 0 and 1, because of this the fuzzy logic component of CI may be a useful resource to the software engineering process in cases concerning the incompleteness of data.  Probabilistic Methods could also be used to improve the software engineering process as they aim to evaluate outcomes of systems mostly defined by the randomness of the real world. Probabilistic methods define the possible solutions based on prior knowledge - this is extremely useful for engineers when working with a client.

From what we have discussed above, It is clear that both AI and CI provides useful methodologies and algorithms to the software engineering process.

## Ethics concerns surrounding this kind of analytics

When measuring the software engineering process -  a single metric that doesn't raise either ethical or problematic concerns may be hard to come by. It particularly difficult to measure because, as I have discussed above, there doesn't seem to exist one reliable, objective metric of developer productivity. Therefore, developers must put up with intrusive or distracting methods of data collection in order for companies to gain the best insight into the Engineering Process. However for engineers and the organisations wanting to measure their productivity the two biggest concerns when it comes to ethics would be ensuring privacy and security.

## Privacy

Through legislation and good business practices, desire for data privacy has expanded to include the software development process itself. As so much consumer data is now stored by organisations we need to question do software engineers require access to this information? What about a simple production system failure? What about system test data that is currently a sampling of the data in production? How can software problems be resolved without the data that caused them to malfunction? How can systems be tested without representative test data? We could use Data Masking ( Data that is Masked, Jumbled, Encrypted, Scrubbed, or even Sanitized.) This is the process used to make the sensitive data void of its real world meaning, but keep it so it is still useful in software testing.

It is not only data collected from clients that may be sensitive, The privacy of the engineer is just as important in the software engineering process. It seemed to be threatened by HackyStat toolkit described earlier. Apparently when Hackystat came into use first developers complained about the automated nature of the data collection, and were unhappy with the invasion of their privacy necessary to gain insight into their workflow. Software engineers didn't want to install software that would collect data about their activities and without their knowledge of what was being submitted. It is understandable that developers would be worried that certain information collected may not work in their favour and may be reason for them being let go or otherwise being punished by the organisation they work for.

## Security

Another liability of more advanced forms of data collection is guaranteeing the security of private information. The more data being collected about employees and clients, the more sensitive data there is to be leaked by an organisation.

Some of the metrics we looked at above involved almost monitoring an engineers every move while working, this means an organization would be storing massive amounts of detailed information about workers actions. This information could be deemed attractive to third party organisations for research, statistical  and other purposes - leaving the system storing the data vulnerable to exploitations such as injections or cross-site scripting(XSS.) This data could then be used or sold by a third party completely unknowns to the staff force of an organisation.

For this reason it is so important to have a proper, robust security system in place. It also may be necessary to take a look at the data being collected and stored by the organisation - perhaps questioning the absolute necessity of said data.

## Conclusion

Ultimately we must figure out how to tie the activities of development teams to the success of the overall organisation, and focus on measuring and improving those related activities. One will never find a single golden set of unchanging metrics, but the search for measurements that are appropriate for your current situation has value in itself. Also its important for organisations to reflect on what they learn over time from these measurements. It’s one thing to set a goal for developers but if you don’t ever go back and look to see not only how you did against that goal but what you learned in the process, you will waste a lot of time and resources. And also - to always have integrity when it comes to the privacy and security of both the client and the engineer.

Sources:

*https://www.softwaretestingmaterial.com/sdlc-software-development-life-cycle/*

*https://softwareengineering.stackexchange.com/questions/275288/what-do-design-and-implementation-mean*

*https://en.wikipedia.org/wiki/Software_maintenance*

*https://medium.com/omarelgabrys-blog/software-engineering-software-process-and-software-process-models-part-2-4a9d06213fdc*

*https://web.cs.dal.ca/~hawkey/3130/SEBackground4.pdf*

*https://en.wikipedia.org/wiki/Software_development_process*

*https://medium.com/@yupyork/the-best-developer-performance-metrics-6295ea8d87c0*

*https://arxiv.org/ftp/arxiv/papers/1805/1805.09485.pdf*

*https://www.computersciencedegreehub.com/faq/what-is-computational-intelligence/*