Measuring Engineering

Emma Bourke | Software Engineering | 24/11/2017



# Introduction

This paper is an introduction into 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. I will be detailing the different methodologies of software metrics and the reasons organizations utilize these. Before I can discuss the measurement of engineering I must explain the process behind software development. It is important to understand the stages of development which metrics investigate. Inevitably there is a large amount of data collected when evaluating software engineering, in the paper I discuss this further.

# Software Engineering Process

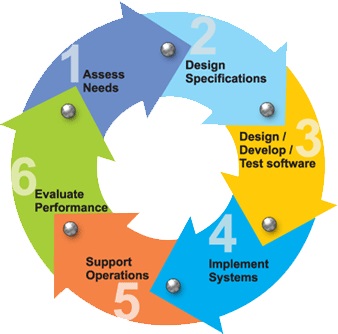
To be able to consider the different ways in which the software engineering process can be measured and assessed, we must first look at what the software engineering process is.

A software development process, also known as life cycle, is a structure imposed on the development of a software product. There are several models and methodologies which each describe certain approaches to a variety of tasks and activities that need to take place during the process. The most successful software development organizations implement process methodologies.

One of the leading models is known as the Capability Maturity Model. Independent reports and assessments are used to grade organizations on how well they create software according to how they implement and execute their processes.

Software Engineering processes are composed of many tasks and activities, it can be a long and tedious process. The nature of developing software means engineers can be working on the same product for long periods of time. The process includes the application of a disciplined approach to many aspects be it design and development or operation and maintenance of the software.

A scale of six process maturity levels have been established which outline the different steps that are imperative to any software engineering process and aid in successful development.

These five steps are as follows:

1. Requirements analysis
2. Design Specification
3. Develop/Unit testing
4. System testing
5. Maintenance
6. Evaluate Performance

Requirements Analysis

• The first phase of the process of requirements and analysis is essential and extremely crucial to the process. Finding out the requirements for the desired software is task number one and is important to interact with the customer and understand what their wants and needs are. Customers often believe they know exactly what the software is to do and how they want to do it. However, this may require skills and experience in software engineering is needed to recognize incomplete, ambiguous or contradictory requirements. Combining the vision of the customer and the skills and knowledge of the software engineer is essential to build a successful product.

Design Specification

• The designing specifications of the software with its architecture, interface design is another key step to software design and engineering. It is also the task of precisely describing the software to be written, in a mathematically rigorous way. This may be to fine tune an already well-developed software. The architecture design refers to the abstract representation of the system. This is crucial in making sure the software meets all requirements of the product decided in phase one. It also should allow for new requirements in the future to be easily made leading to a good development process.

Design/Develop and Testing

• The next step is to begin implementation and design the software to then begin testing the system. This phase is another important process which will eventually decide on the workings of the software. In this phase, all the design aspects are reduced to code by the team of software engineers assigned to the project. This involves engineers working together over long periods of time. Through-out this phase testing of parts of the software will take place, this is very important as many software engineers working on the same project may lead to error in combined code. It is not necessarily the largest portion of time but it is one of the rigorous steps in the process.

Support Operations

• Maintenance is one of the final stages of the software engineering process and is imperative for the successful and constant running of the software without errors. Maintaining and enhancing software to cope with newly discovered problems or new requirements can take far more time than the initial development of the software. Not only may it be necessary to add code that does not fit the original design but just determining how software works at some point after it is completed may require significant effort by a software engineer.

Performance Evaluation

• Evaluation of performance is the final stage of the process and one which is very important for future projects and development as it allows time for engineers to reflect and improve. Documentation is also a very important task in the development of the process and the overall evaluation. Documentation of the internal design of the software is done for future maintenance and enhancement. Most maintenance is extending systems to do new things, which in many ways can be considered new work.

# Capability Maturity Model

One of the leading models is known as the Capability Maturity Model. The CMM was developed by the SEI in Pittsburgh, US and is used extensively for many important software and government projects all over the world. The CMM is a way of managing the software processes of an organization.

It helps to develop and refine organizations processes. The model defines five maturity levels which all must be met when completing a development project, they are:

Level 1. - Initial

Level 2. - Repeatable

Level 3. - Defined

Level 4. - Managed

Level 5. - Optimized

The five process levels are used as a scale to assess organizations and rank them according to the standardization of processes in the specific area being assessed.

There are many diverse areas that need to be measured throughout software development projects such as software engineering, systems engineering, project management, risk management, system acquisition, information technology (IT) services and personnel management.

Maturity Model

A Maturity Model is a structured collection of elements that describes the characteristics of effective processes. This model provides a place to start when measuring data. It also has the benefit of a community’s prior experiences and is presented in a common language and a shared vision. It establishes a framework for prioritizing actions and a way to define what improvement means for your organization.

The model can be used as a benchmark for assessing different organizations for equal comparison. It shows the growth of the company based upon the project the company is dealing with and the clients

Key Process Areas

The nature of software development and the fact that each project is unique and comes with its own specifications means that there can be an array of processes to complete. Within each of the maturity levels there are key process areas which characterize that level and for each key process there are five definitions identified which are:

1. Goals

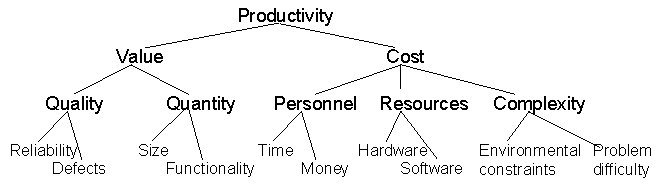
2. Commitment

3. Ability

4. Measurement

5. Verification

These are the stages that organizations must go through on the way to becoming mature. These different stages are assessed by authorized lead assessors to make sure the organizations development teams are fulfilling their roles in every area, this then reflects on the overall productivity level.



# Software Engineering Metrics

The nature of the software engineering process means it can take a large amount of time for software engineers to complete a project and build a product or system. This process must be monitored constantly to be able to assess the progress and productivity throughout the life cycle of a project. Most software development companies use software engineering metrics to quality check and assess performance of employees.

Software Measurement and Metrics

A Measurement is an indication of the size, quantity, amount or dimension of a specific attribute of a product or process. For example, the number of errors in a system is a measurement.

A Metric is a measurement of the degree that any attribute belongs to a system, product or process. For example, the number of errors per person hours would be a metric.

To assess the quality of the engineered product or system and to better understand the models that are created, some measures are used. These measurements are then collected, converted into metrics and documented throughout the development process with the intention of improving the software process on a continuous basis.

Metrics is often interchangeable with measure but it is important to know the difference. Measure can be defined as quantitative indication of amount, dimension, capacity, or size of product and process attributes. Measurement can be defined as the process of determining the measure. Metrics can be defined as quantitative measures that allow software engineers to identify the efficiency and improve the quality of software process, project, and product.

Why?

There are many reasons why measures are taken. Software is measured to establish the quality of the current product or process and to predict future qualities. It also is used to improve the quality of a product or process and to determine the state of the project in relation to budget and schedule.

Measurement helps in estimation, quality control, productivity assessment and project control. Measurement also allows software engineers to gain an insight into the different project methods which work and the development methods of certain projects.

# Measurable Data

What data is relevant and measurable when measuring the development of software engineering?

Before data is collected and used it is necessary to know the type of data involved in software metrics.

The table below shows different types of data in metrics along with their description and the operations that can be performed on them.

**Type of Data Measured**

|  |  |  |
| --- | --- | --- |
| **Type of data** | **Possible operations** | **Description of data** |
| Nominal | =,≠ | Categories |
| Ordinal | <, > | Ranking |
| Interval | +, - | Differences |
| Ratio | / | Absolute zero |

Nominal data: Data in the program can be measured by placing it under a category. The category of program can be a database program, application program, or an operating system program. For such data, operation of arithmetic type and ranking of values in any order is not possible. The only operation that can be performed is to determine whether one program is the same as another program.

Ordinal data: Data can be ranked according to the data values. For example, experience in a certain application domain can be rated as very low, low, medium, or high. Thus, experience can easily be ranked according to its rating.

Interval data: Data values can be ranked and substantial differences between them can also be shown. For example, a program with a complexity level of 8 is said to be 4 units more complex than a program with complexity level 4.

Ratio data: Data values are associated with a ratio scale, which possesses an absolute zero and allows meaningful ratios to be calculated. For example, program lines expressed in lines of code.

# Computational Platforms

Large software organizations must collect useful data to be able to build successful software. To provide useful data, a data collection methodology must display certain attributes. Most of the data collected by organizations is collected by the test phase. Therefore, completion of analysis must await completion of engineering. However, for accuracy, it’s important to collect data and validate con-currently with development.

In a reasonable well-controlled software development environment documentation and code are placed under some form of configuration control before being released to their users. Changes can then be made to the baseline design code and not have to go through full configuration control. Careful validation and specification of data is key for consistency of data measurement.

The following summarizes the data attributes in relation to the study of projects from software production environments involving teams of software engineers.

1. The data must contain information of the types of errors and changes made
2. The data must include the cost of making the changes
3. Data to be collected must be defined specified to the goals laid out at the start
4. Data should include studies of projects from production environments involving teams of programmers
5. Data analysis should be historical and collected con-currently with development
6. Data classification schemes must be carefully specified for the sake of repeating the analysis

Design and Test Data Collection Form

To provide a platform for a permanent copy of the data and to reinforce programmer’s memories, a data collection form is used. Forms must be constructed so that they can be used to answer the questions of interest laid out at the start of the software measurement process. You need to take into consideration different programming environments and allow for flexibility in describing the different errors and then corresponding changes made to correct these.



These forms can be used to collect a range of data.

* Process mapping is recorded by helping define the project process, depict inputs, outputs and units of activity. An example of this is recording the frequency of commits in Github, a control repository hosting service. It can serve as an instruction manual or a tool for facilitating detailed analysis and optimization of workflow and service delivery.

Size projections are used to collect code size and level of progress.

**Source byte size** - a measure of the actual size of the source code data. The overall byte size of the project can be estimated which would give a better indication of the type/size of hardware needed to use the software. However, the byte size of the source code would vary greatly depending on which programming language was used to develop it. For example, a program developed in Java could be much smaller than the same program coded in COBOL.

**Source lines of code** - SLOC gives the size of the project by counting all the lines of source code in a project. It is used to measure effort both before as an estimate and after as an actual value. Combined with time and date of work we can assess the productivity of the programmer. When code is written, integration and unit testing can be performed so measures of programming productivity and quality can be assessed.

* Productivity projections
* Cost estimation

Where does all the collected data go?

Software Engineering data in the real world. There are many leading companies such as Accenture and Ernst and Young, which store, sort and analyze huge volumes of engineered data. This data is used to create an array of models and algorithms which are then used to answer specific queries and help make decisions about future software developments or find solutions for problems with current projects.

Accenture

Accenture is a global management consulting and professional services company that provides a strategy, consulting, digital, technology and operations services. Data Analytics is a major area within Accenture.

Accenture work at bridging the gap between technology and business to help software companies understand their employee’s requirements to improve performance, creating better products, work more cost-efficiently and become more productive overall. The Accenture platforms is a fast, flexible and scalable-ready to transform performance in weeks, not years. They intend to simplify analytics for businesses with cloud-based insights with the use of a range of analytics applications. These applications take in client’s data and each app manages the complete end-to-end process, providing organizations with immediate access to the tools needed to make data-driven decisions and solve complex business issues.



Code Climate

Code Climate is a platform for organizations to take control of their code quality by incorporating fully configurable test coverage and maintainability data throughout the development workflow. It allows organizations to build a database and then look at certain trends in productivity and code quality. They can then relate this to different variables like specific times of the day or areas of the building where code quality is better or worse. Companies can build trend graphs and statistical models. Using this data companies can find the root of certain problems, investigate further and come up with a solution.



# Ethical Concerns and Mismeasurement

There are many issues which arise when measuring and assessing software developers’ work. This is constantly a problem for software organizations as specific software projects and programs are highly sensitive and have the potential to generate a lot of revenue for both the client and developer. The main problems are the ethical concerns and the mismeasurement of individuals work.

* + - The concerns about privacy and copy-cat code are problems which have arisen in last decade due to the development of the web and the difficulty of keeping things private. There have been many lawsuits against companies for the way in which they measure and assess their developers and the ethical concerns behind this.
* The potential of mismeasurement is also a main issue, it is extremely difficult to assess the progress of software engineering due to the individual work environment and length of the process. Collecting misleading data is one of the main reasons why companies run into problems later in the process. Creating statistical models with misleading data can make finding solutions to the problem very difficult.

“Measurement is the empirical object assignment of numbers according to a rule derived from a model or theory to attributes of objects or events with the intent of describing them” – Cem Kaner.

“The map is not the territory” - Alfred Korzybski.

There are two main reasons for mismeasurement, map-territory error and reification error. A common error is the confusion of a model with reality. Mathematical or simulation models may help us to understand a system or situation but real life always differs from the model in some shape or form. However, just because a model is wrong doesn’t mean it isn’t useful. A reification error occurs when an abstract idea is treated as if it were a physical object or real event. Counting abstract ideas is dangerous and comparing these counts is even more dangerous. For example, if one software developer comes up with 6 ideas that week and another generates 10 ideas, are they branded the better engineer? How does one know which ideas are better or more importantly feasible? Measuring intelligence, performance, commitment, motivation, leadership and the likes is extremely difficult and must be dealt with accordingly. For example, measuring performance on a scale of 1-10 which is often seen in reviews. It is hard to believe an entire span of thousands of hours of somebody’s effort can be boiled down to a single number or word. Yet many metric programs do this all the time. We cannot just say some one has a bad attitude, there must be tangible evidence, evidence we can count and measure.

In the software industry, everyday developers attempt to count requirement and defects, they then divide them into defects per thousand requirements or defects per lines of code. They forget that these are all ideas or concepts. What one person sees as a defect another may not have any problem with at all.

Researchers conducting studies of software engineering often face ethical dilemmas. As software takes over more of our lives, the ethical ramifications of decisions made by programmers only become greater. The issue constantly arises due to the fact the community have yet to adopt a common set of guidelines geared towards ethical issues.

The most relevant area of discussion within the ethics issues are the researcher projects which employ human subjects or involve the collection of information that leads to the identification of individuals. When observing individuals, a range of information can be collected. The examination of artifacts relating to the research such as source code and documentation can also be under observation. Whether this involves metrics, workplace studies or process studies its irrelevant, this type of examination can be both invasive and damaging. The software community need to concern themselves as researchers who upset their subjects risk losing their co-operation or honesty. They can lose their access to developers and the necessary funding needed to carry out the research. If employees are coerced into participating this can lead to invalid data being collected.

In my opinion, formal guidelines should be drafted together so that those researching are aware of the code of ethics before starting any form of observation or collection of data. Subjects should also understand their rights as developers and employees to ensure that they are appropriately shielded from harm such as loss of employment.

# Conclusion

I believe there are both advantages and disadvantages with conducting such measurement of software. It depends on a series of variables such as the individual themselves, their environment and the type of project they are working on. Software measurement and metrics help us a lot, to evaluating software process as well as the software product. The set of measures, some identified in this paper, provide the organization with better insight into the validation activity, improving the software process towards the goal of building a successful software while having an efficient management process and being aware of ethical concerns. Well-designed metrics with documented objectives can help an organization obtain the information it needs to continue to improve its software product, processes, and customer services. Therefore, I think future research is needed to extend and improve the methods used. This then allows use to use metrics that have been validated on one project to valid measures of quality on future software projects. It is important for researchers to consider the ethical issues raised by their project while it is still in the planning stage as to avoid any issues later in the process. From my research, I believe the difficulties involved in conducting large-scale controlled software engineering measurement have yet prevented successful evaluations of software development methods. It is important however to consider different environments and situations. Software engineers should also use less formal techniques that can be used in real working environments to establish long-term trends.

# Bibliography

* <http://www.selectbs.com/analysis-and-design/what-is-a-software-development-process>
* <http://www.selectbs.com/process-maturity/what-is-the-capability-maturity-model>
* https://codeclimate.com/
* <https://www.cs.umd.edu/~basili/publications/journals/J23.pdf>
* <https://en.wikiversity.org/wiki/Software_metrics_and_measurement#Step_3._Specify_the_Metrics_needed_to_answer_the_questions>.
* <http://ecomputernotes.com/software-engineering/software-measurement>
* <https://www.cs.umd.edu/~basili/publications/journals/J23.pdf>
* <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.134.712&rep=rep1&type=pdf>
* <https://www.accenture.com/ie>
* <https://www.infoworld.com/article/2607452/application-development/12-ethical-dilemmas-gnawing-at-developers-today.html>