Measuring Software Engineering CSU33012

Caroline Liu

7th November

Contents

1 Abstract

This report discusses 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.

2 Introduction

The IEEE defines software engineering as:

- The application of a systematic, disciplined, quantifiable approach to the development,operation and maintenance of software; that is, the application of engineering to software.
- The study of approaches as in the above statement.[1]

In summary, software engineering is the development of software fulfilling the users needs and thus using well-defined scientific principles, methods and procedures to execute the process.

The main principles of software engineering include:

- **Separation of concerns**: when specifying the details for the development of an application or its component, there are often two concerns that need to be encountered: basic functionality and support for data integrity. Run-time efficiency is a common example of this.
- **Modularity**: this is a specialisation of the above principle where its procedure is to separate the software components according to functionality and components.
- **Abstraction**: this is another specialisation of the principle of separation of concerns. This requires for the developer to separate the components of the software from their implementation and considering them by functionality and implementation.
- **Anticipation of Change**: working out an automated solution is both a learning experience for the users and the developers. When problems arise, the users would expect the developers to come up with a reasonable solution as soon as possible. This

1. ACM(2007), *Computing Degrees and Careers*, `http://computingcareers.acm.org/?page_id=12`.

would not only be a timely challenge, but also one that requires both the user and developer to be familiar in their own perspective about the program and the program itself to have been developed in such a manner that its components and other functionalities would not be prone to the changes being made.

- **Generality**: this is closely related to the principle of anticipation of change. This states that it is key when designing the software to ensure that it is free from unnatural restrictions and limitations. An example of this is the two digit formatting of year numbers, which lead to the "year 2000" problem. Users learn their actual needs when they see the gradual development of the program.
- **Incremental Development**: this is a method that aids in simplifying the process of software development by only considering rigid testing on the incremental component added to the software at each stage until the projected functionalities are fulfilled.
- **Consistency**: this is a principle is a recognition of the fact that it is easier to do things in familiar context serving both the user and the software engineer for future improvements. Coding style is an example of this.[2]

3 History of Software Engineering

Digital computers first appeared in the early 1940s, the instructions to make them operate were wired into the machine.[3] This aided the discovery of the von Neumann architecture of computers. This gradually lead to the establishment the division between hardware and software as the complexity of abstraction being used to deal with computing.

Then came the development of programming languages in the early 1950s such as Fortran, ALGOL, and COBOL to deal with real world problems. Shortly after this, cam the *software crisis* of the 1960s, 1970s, and 1980s, this caused the stimulated the development of software engineering in which some fundamental problems a the times were identified. These included concepts such as:

2. University of Minnesota Duluth, *Principle of Software Engineering*, `https://www.d.umn.edu/~gshute/softeng/principles.html`.

3. Leondes Cornelius T.(2002), *Intelligent Systems: Techonology and Applications, 1.4 Computers and a First Glimpse at Al (1940s), CRC Press. p. I-6. ISBN 978-0-8493-1121-5.*

- Cost and budget overturns
- Property damage
- Life and death[4]

During the 1960s, writing software has evolved into finding the best ways to maximise the quality of software and methods in the building it. Quality can refer to how maintainable software is, to its stability, speed usability, testability, readability, size, cost, security and number of flaws, as well as many more attributes.

Peter G. Neumann has kept a contemporary list of software problems and disasters.[5] The crisis gradually faded from view due to the prolonged period of the instance. Michael A. Jackson has written extensively about the nature of software engineering,[6] framing the problems that whether software engineering is to become an engineering science. During this period also, David Parnas introduced two of the the main concept of software engineering: modularity and data integrity in 1972.

For decades, solving the software crisis was at most importance for researchers and companies producing software tools. The cost of owning software was the burden of the time.

Every new technology and practice including tools, discipline, formal methods, process and professionalism, from the 1970s through the 1990s was seem as a *silver bullet* to solve the software crisis. Though in 1986, an article titled "No Silver Bullet" was published by Fred Books, stating the argument that "*there is no single development, in either technology or management technique which by itself promises even one order of magnitude (tenfold) improvement within a decade in productivity, in reliability, in simplicity.*"[7]

In 1984, the Software Engineering Institute was established and the SEI Software Process Program was founded and aimed at understanding and managing the software engineering process.

As of today, these components have joint together enabling us to form a more precise understanding of the term software engineering, also allowing us to have a more identified process when developing software.

4. Wikipedia, *Software Engineering*, `https://en.wikipedia.org/wiki/Software_engineering`.

5. RISKS Forum Digest, *The Risks Digest*, `http://catless.ncl.ac.uk/Risks/`.

6. Michael Jackson, *Engineering and Software Engineering*.

7. Frederick P. Brooks Jr., *No Silver Bullet - Essence and Accient in Software Engineering*, `https://www.cgl.ucsf.edu/Outreach/pc204/NoSilverBullet.html`.

4 Measurable Data

Today, data is a way for software engineers to keep track of their delivered processes and keep up with the accelerated industry. Software metrics is a standard way of measuring many activities of production, which include the following:

- **Product metrics**: which describe the characteristics of the products, including size, complexity, design features, and quality level.
- **Process metrics**: this data can be used for future development and maintenance purposes of the program.
- **Project metrics**: this data describes the project characteristics and execution.[8]

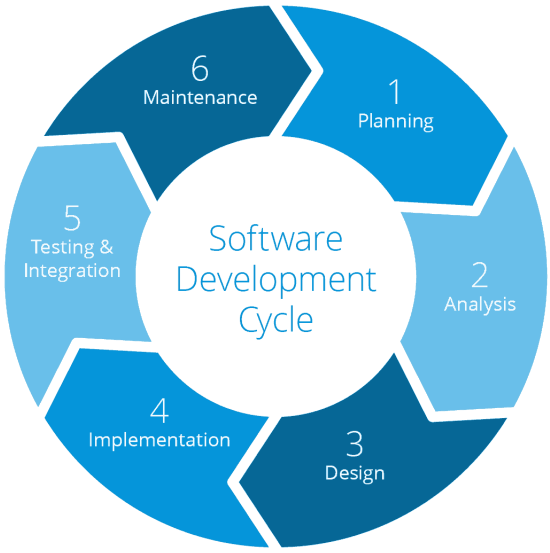
The main concern for tracking software metrics is to make measurements easy to collect or it will not be done. The purpose of collecting such data is to aid the improvements that could be made to the development team for higher production of the team. Gathering of such data should not be a burden of the process. Therefore, taking it into consideration, software metrics should have the following characteristics:

- Relevant and reliable
- Simple and computable
- Consistent and objective
- Independent of programming language or platforms
- cost-efficient
- Adaptable[9]

Hence, it is important that software development platforms that automatically track these metrics.

8. Wikipedia, *Software Engineering*.
9. Olena Herasymchuk, *A Guide to the Top Software Development Metrics That Will Take Your Business to the Next Level*, `https://www.daxx.com/blog/development-trends/top-software-development-metrics`.

4.1 Agile Development Methodology



10

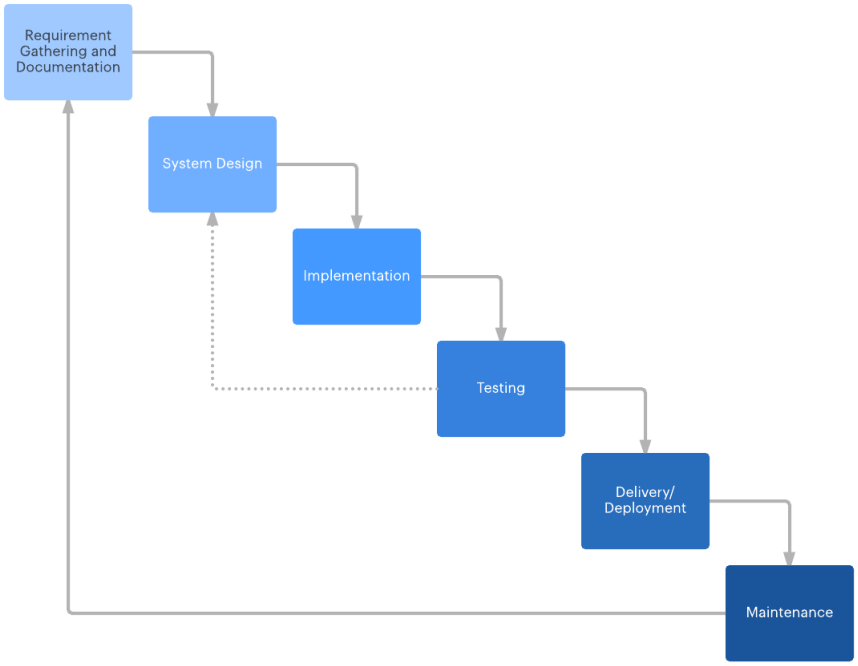
In agile methods, teams develop the software in iterations that contain mini-increments of the new functionality. This is more beneficial than betting everything all at once, but work is consumed in smaller units, robust testing would be done on the currently delivered work showing indications of improvements allowing restart of the mini-process, until the features and functionalities fulfill the current projections of the process. There are many different forms of the agile development method, including scrum, crystal, extreme programming (XP), and feature-driven development (FDD).

Benefits of this type of development methodology is that it allows software to be released in iterations. Iterative improvements on the software would allow the team to identify problems at an early stage of the process an aid in the end quality of the deliverable product.

Agile, however, does have it's disadvantages including more time for the team invested into communication and more commitment required by the individual. It also has less predictability and would easily fail once the progress is off-track.

10. Jasper can der Hoek, *Pursuing a Full Agile Software Development Life Cycle*, `https://www.mendix.com/blog/pursuing-a-full-agile-software-lifecycle/`.

4.2 Waterfall Method



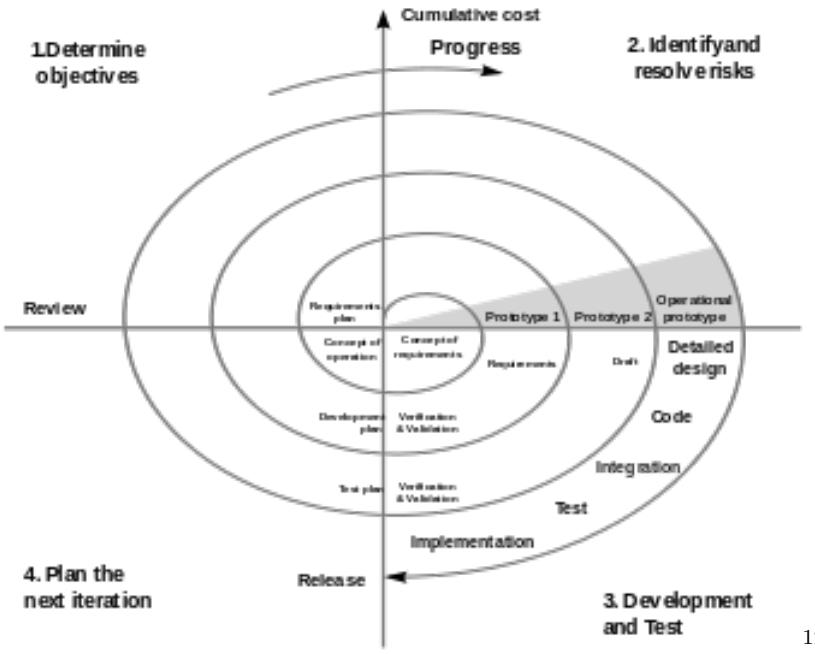
Waterfall method is a rigid linear model that consists of sequential phases (requirements, design, implementation, verification, maintenance) focusing on distinct goals, which is the most traditional software development method. Each phase must be fully complete before proceeding. It is not quite flexible in that there is usually no process for backtracking previous phases.

Benefits of this method include a clear, well defined structure of procedures and the clear set a stable goal from the initial stages of development. This make it easier to understand and manage the overall project.

Disadvantages are obvious as this method does not allow for backtracking of processes, it is only when one stage is fully complete, that allows the team to move on to the next stage and delays testing.

11. Lucidchart Content Team, *What the Waterfall Project Management Methodology Can (and Can't) Do to You*, `https://www.lucidchart.com/blog/waterfall-project-management-methodology`.

4.3 Spiral Method



12

The spiral method adapts the spiral model, which is one of the most important software development life cycle model. It is graphically represented as infinite many loops on a spiral, the number of loops depend on the individual project. Each loop is called a phase of the process. It allows a more dynamical development of the project. The radius of the spiral at any point represent the expenses of the project and the angular dimension represents the progress made so far in the current phase.

This method is good for large project and allows for risk analysis and risk handling at every phase. Changes in the requirements are allowed in later phases of the process and can be incorporated accurately by using this method.

This method is much more complex than others and costly, therefore not suitable for small project. It is also too much dependable on risk analysis, without very highly experienced expertise, it is going to cause failure in the process. Management in time is also an issue as the time consumed by each phase at the initial stage of the project is unknown.

12. Wikipedia, *Spiral Model*, `https://en.wikipedia.org/wiki/Spiral_model`.

4.4 Benefits of Tracking and Analysing Software Metrics

Benefits of tracking and analysing software metrics is to determine the quality of the current product or process, and give an indication of the possible improvements that can be made to the software project, thus measure the quality of the overall completed project. This also aids the productivity in the workplace, for instance, managers would be able view this data and identify, prioritise, track and communicate or any other issues that can be resolved, benefiting the production levels.

5 Computational Platforms

Computational Platforms allows the above development methods to be assessed and software to be executed. Computing platforms have different abstraction levels, including a computer architecture, an OS, or run-time libraries.[13] A platform can be seen as a constraint but also assure better code qualities and find "bugs" and aid the developer to find better solutions. The process in reviewing code could be time consuming but allows for better quality code.

Platforms serve various purposes including allowing for automated execution of code, collaboration and development on cloud servers.

5.1 Cloud Server

Most services run off servers as opposed to personal computers, therefore there are platforms and tools built to deploy and execute the scripts and maintain the server. Examples of these include Docker, Microsoft Azure and Google Compute Engine.

5.2 Collaboration and Integrity Checking

The main output of the process of software engineering is the deliverable product. Nowadays, projects are often based on teams, and therefore require a collaboration platform for developing methods such as agile to work out. Platforms such as Github allow individuals to work by themselves and eventually merge their work into the main project, whilst also keeping backup and tracking their work. Features of such platforms are far more than this.

13. Free On-line Dictionary of Computing, *platform*, `http://foldoc.org/platform`.

6 Algorithmic Approaches

There are various ways to measure and analyse the software engineering process. One of the most effective ways would be to use machine learning as it allows the process to develop for itself a set of test and rules dynamically.

6.1 Machine Learning

Machine Learning defined by Tom Mitchell in 1998, is a well posed learning problem where a computer program is said to learn from the experience E, with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E.[14]

Machine learning today, unlike the past, enable the program to learn from previous computations to produce reliable, repeatable decisions and results rather than before, where it initialised as pattern recognition and the theory that computers can learn without being programmed to perform specific tasks.[15] Widely publicised examples of machine learning would include the following:

- The self-driving Google car
- Amazon/Netflix recommendations
- Analysing customer reviews on social media
- Fraud detection

Machine learning is important important as of today, data is of the essence. Analysis and automation of theses analyses to deliver faster and more accurate results allow companies to gain insights into their own performance against others of similar functionalities in the industry.

Here are some of the various types of machine learning:

14. Tom Mitchell, *Machine Learning*, `http://profsite.um.ac.ir/~monsefi/machine-learning/pdf/Machine-Learning-Tom-Mitchell.pdf`.
15. SAS, *Machine Learning*, `https://www.sas.com/en_ie/insights/analytics/machine-learning.html`.

6.1.1 Supervised Learning

These types of algorithms are trained with the "right answers" given. These algorithms mainly deal with regression problems, where it is to predict continuous valued output, and classification in which the output is discrete i.e. yes or no.

6.1.2 Unsupervised Learning

Theses algorithms are trained without telling it the "correct" answer. It itself must analyse the data and find some sort of structure within. These algorithms are catagorised into two parts: clustering, where the aim is to discover the inherent groups of data, and association, where the aim is to discover rules that describe large portions of the data.

6.1.3 Semi-supervised Learning

These types of algorithms fall in between unsupervised and supervised learning. It typically takes in a small amount of labeled data with a large amount of unlabeled data, which study has shown it improves the learning accuracy.

6.1.4 Reinforcement Learning

This is an area where is it mainly about taking suitable actions to minimise rewards in a particular situation, which are often seen in gaming, robotics and navigation areas. The training of these algorithms are based upon the input (the initial state of the model) and the user will decide to reward or punish the model base on the output. The learning proceeds and the optimal solution is decided based on the maximum reward.[16]

7 Ethical Concerns

Ethical issues occur when a given decision, scenario or activity creates a conflict with society's moral principles. Software engineers are to respect and be aware of the ethical concerns that may arise from their code and actions in the workplace. Some of the ethical issues in software engineering include:

- **Privacy**: handling, storing, sharing user data only under circumstances and purposes that the user agrees to

16. Geeks for Geeks, *What is Reinforcement Learning*, `https://www.geeksforgeeks.org/what-is-reinforcement-learning/`.

- **Transparency**: transparent decision-making procedures in intelligent systems, publicly available ethics policies by software development organisations.
- **Diversity**: gender, race, and age distribution of professionals in a development team
- **Common Goods**: contributing to, using, promoting open source software
- **Work Ethics**: decisions on the priority of implementations of requests before the product is released
- **Accountability**: deciding the responsibility when flaws occur[17]

A fine example would be the General Data Protection Regulations, GDPR, brought out to give EU citizens more control over the their personal data. The aim is that both the individuals and companies would benefit, as companies must measure their performance under a legal basis.[18]

It is hard to measure the rights and wrongs on tracking data as to the different ethical limits for different societies and individuals. It is certain that there is still a long way to go as to agreeing upon these regulations.

8 Conclusion

This report discusses the various ways of measuring and assessing the process of software engineering with the components of measurable data, computational platforms, algorithmic approaches and ethical issues, that all aid the process to become more efficient and society-friendly.

References

ACM(2007). *Computing Degrees and Careers*. `http://computingcareers.acm.org/?page_id=12`.

17. Research Gate, *Examples of Ethical Issues in Software Engineering*, `https://www.researchgate.net/figure/Examples-of-ethics-issues-in-software-engineering_tbl1_326045987`.

18. Wikipedia, *General Data Protection Regulation*, `https://en.wikipedia.org/wiki/General_Data_Protection_Regulation`.

Computing, Free On-line Dictionary of. *platform*. http://foldoc.org/platform.

Cornelius T.(2002), Leondes. *Intelligent Systems: Techonology and Applications, 1.4 Computers and a First Glimpse at Al (1940s), CRC Press. p. I-6. ISBN 978-0-8493-1121-5.*

Digest, RISKS Forum. *The Risks Digest*. http://catless.ncl.ac.uk/Risks/.

Gate, Research. *Examples of Ethical Issues in Software Engineering*. https://www.researchgate.net/figure/Examples-of-ethics-issues-in-software-engineering_tbl1_326045987.

Geeks, Geeks for. *What is Reinforcement Learning*. https://www.geeksforgeeks.org/what-is-reinforcement-learning/.

Herasymchuk, Olena. *A Guide to the Top Software Development Metrics That Will Take Your Business to the Next Level*. https://www.daxx.com/blog/development-trends/top-software-development-metrics.

Hoek, Jasper can der. *Pursuing a Full Agile Software Development Life Cycle*. https://www.mendix.com/blog/pursuing-a-full-agile-software-lifecycle/.

Jackson, Michael. *Engineering and Software Engineering*.

Jr., Frederick P. Brooks. *No Silver Bullet - Essence and Accient in Software Engineering*. https://www.cgl.ucsf.edu/Outreach/pc204/NoSilverBullet.html.

Minnesota Duluth, University of. *Principle of Software Engineering*. https://www.d.umn.edu/~gshute/softeng/principles.html.

Mitchell, Tom. *Machine Learning*. http://profsite.um.ac.ir/~monsefi/machine-learning/pdf/Machine-Learning-Tom-Mitchell.pdf.

SAS. *Machine Learning*. https://www.sas.com/en_ie/insights/analytics/machine-learning.html.

Team, Lucidchart Content. *What the Waterfall Project Management Methodology Can (and Can't) Do to You*. https://www.lucidchart.com/blog/waterfall-project-management-methodology.

Wikipedia. *General Data Protection Regulation*. https://en.wikipedia.org/wiki/General_Data_Protection_Regulation.

———. *Software Engineering*. https://en.wikipedia.org/wiki/Software_engineering.

———. *Spiral Model*. https://en.wikipedia.org/wiki/Spiral_model.