Measuring Engineering

CS3012

Michael McGuinness
16322635

3rd November 2019

Table of Contents

Abstract

This report discusses the ways in which the software engineering process can be measured and assessed in terms of measurable data, the computational platforms available to perform this kind of work, the algorithmic approaches available and the ethics surrounding this kind of analytics.

Introduction

Software engineering is the process of analyzing user needs and designing, constructing, and testing end user applications that will satisfy these needs through the use of software programming languages. It is the application of engineering principles to software development.
The main principles of engineering are testability, maintainability, integrity, external integration, ethics and management.
Testability - It is designed to have a repeatable test that can be performed to ensure that it is as expected.
Maintainability - It is designed to last.
Integrity - It is designed to have structural integrity. In memory state must not just become corrupt.
External integration - It is designed to have a particular well defined way to integrate with its environment without any confusion.
Ethics - Act primarily in the best interests of:

A. the above principles
B. the users
C. the client

Management - An engineer is proactively involved in the management of various responsibilities because the ultimate responsibility rests on the engineer. An engineer is actively involved in listening to specialists and will let them do their speciality but is proactive in coordinating the various specialists towards the ultimate goal.

History of Software Engineering

From its beginnings in the 1960s, writing software has evolved into a profession concerned with how to best create software and maximise it quality. Quality can refer to the maintainability, stability, speed, usability, testability, readability, size, cost, security, and number of flaws of the software, as well as to less measurable qualities like elegance, conciseness, and customer satisfaction.
How best to create high quality software is a problem covering software design principles, so-called "best practices" for writing code, as well as broader management issues such as optimal team size, process, how best to deliver software on time and as efficiently as possible, work-place "culture", hiring practices, and so forth. All this falls under the broad rubric of software engineering.

Putative origins for the term software engineering include a 1965 letter from ACM president Anthony Oettinger, lectures by Douglas T. Ross at MIT in the 1950s. The NATO Science Committee sponsored two conferences on software engineering in 1968 and 1969, which gave the field its initial boost.
Software engineering was spurred by the so-called software crisis of the 1960s to 1980s, which identified many of the problems of software development. Many projects ran over budget and schedule, some caused property damage and loss of life. What was originally defined as a productivity issue became a quality issue too.
The OS/360 operating system was a classic example of cost and budget overruns. It produced one of the more complex software systems of its time, however, a few fundamental mistakes were made which cost multiple millions of dollars, including the lack of developing a coherent architecture before starting development.
Meanwhile, software such as Therac-25 is a classic example of an incident of how software defects can kill.
Over the last 10–15 years Michael A. Jackson has written extensively about the nature of software engineering, has identified the main source of its difficulties as lack of specialization, and has suggested that his problem frames provide the basis for a "normal practice" of software engineering, a prerequisite if software engineering is to become an engineering science.

People were looking for issues, and many of the solutions people had were considered silver bullets. Many tools, standards and documentation were considered silver bullets. Tools include structured programming and object-oriented programming. Some argued that programmers lacked discipline. Some believed that if formal engineering methodologies would be applied to software development, then production of software would become as predictable an industry as other branches of engineering.

The use of defined processes and methodologies like the Capability Maturity Model were considered a good solution. The level of professionalism, licenses and a code of ethics was worked on.
In 1986, Fred Brooks published his No Silver Bullet article, arguing that no individual technology or practice would ever make a 10-fold improvement in productivity within 10 years. Debate about silver bullets raged over the following decade, people still often refer to their process or technology as the silver bullet, the thing that will solve software engineering. However, so far each new thing has made incremental improvements to productivity and quality. Today, these have built on top of each other and the quality of work produced in less time is strides ahead of the past.

Measurable Data

These days data is very important. A software engineering workplace involves rapid development and delivery with constantly changing requirements. This makes it hard to track progress and improvement. In order to be productive and keep up with the industry, teams need to track their performance in terms of different projects. Questions concerning completion time, effort needed, quality of code and costs need to be considered and answered. Thus teams need to have measurable data to track. There have been several methods that have developed to track performance of a team.

Metrics

Management Metrics

- Size: Lines of Code (LOC), Thousand Lines of Code (KLOC) - Even though the LOC metric is widely used, using it comes with some problems and concerns: different languages, styles, and standards can lead to different LOC counts for the same functionality; there are a variety of ways to define and count LOC– source LOC, logical LOC, with or without comment lines, etc.; and automatic code generation has reduced the effort required to produce LOC.
- Size: Function points, Feature Points
- Individual Effort: hours
- Task Completion Time: hours, days, weeks
- Project Effort: person-hours
- Project Duration: months
- Schedule: earned value
- Risk Projection: risk description, risk likelihood, risk impact

Software Quality Metrics

- Defect Density - Defects/KLOC (e.g., for system test)
- Defect Removal Rate – defect removed/hour (for review and test)
- Test Coverage
- Failure Rate

Software Requirements Metrics

- Change requests received, open, closed
- Change request frequency
- Effort required to implement a requirement change
- Status of requirements traceability
- User stories in the backlog

Software Design Metrics

- Cyclomatic Complexity
- Weighted Methods per Class
- Cohesion - Lack of Cohesion of Methods
- Coupling - Coupling Between Object Classes
- Inheritance - Depth of Inheritance Tree, Number of Children

Software Maintenance and Operation

- Mean Time Between Changes (MTBC)
- Mean Time To Change (MTTC)
- System Reliability
- System Availability
- Total Hours of Downtime

Methods for Software Development

Many methods have sprung to light in order to improve performance. The metrics talked about are used to assess the performance of a given team using any of these methods. The following methods are some of the most prominent.

Waterfall Model

The waterfall model is a breakdown of project activities into linear sequential phases, where each phase depends on the deliverables of the previous one and corresponds to a specialisation of tasks. The approach is typical for certain areas of engineering design. In software development, it tends to be among the less iterative and flexible approaches, as progress flows in largely one direction through the phases of conception, initiation, analysis, design, construction, testing, deployment and maintenance. It is generally considered outdated and has been replaced by more modern methods such as the agile and spiral methods.

Agile Method

Agile software development refers to software development methodologies centered around the idea of iterative development, where requirements and solutions evolve through collaboration between self-organizing cross-functional teams. The value in Agile development is that it enables teams to deliver value faster, with greater quality and predictability, and greater aptitude to respond to change.

Agile processes generally promote a disciplined project management process that encourages frequent inspection and adaptation, a leadership philosophy that encourages teamwork, self-organization and accountability, a set of engineering best practices intended to allow for rapid delivery of high-quality software, and a business approach that aligns development with customer needs and company goals. Agile development refers to any development process that is aligned with the concepts of the Agile Manifesto.
The core values of agile are:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

The principles articulated in the Agile Manifesto are:

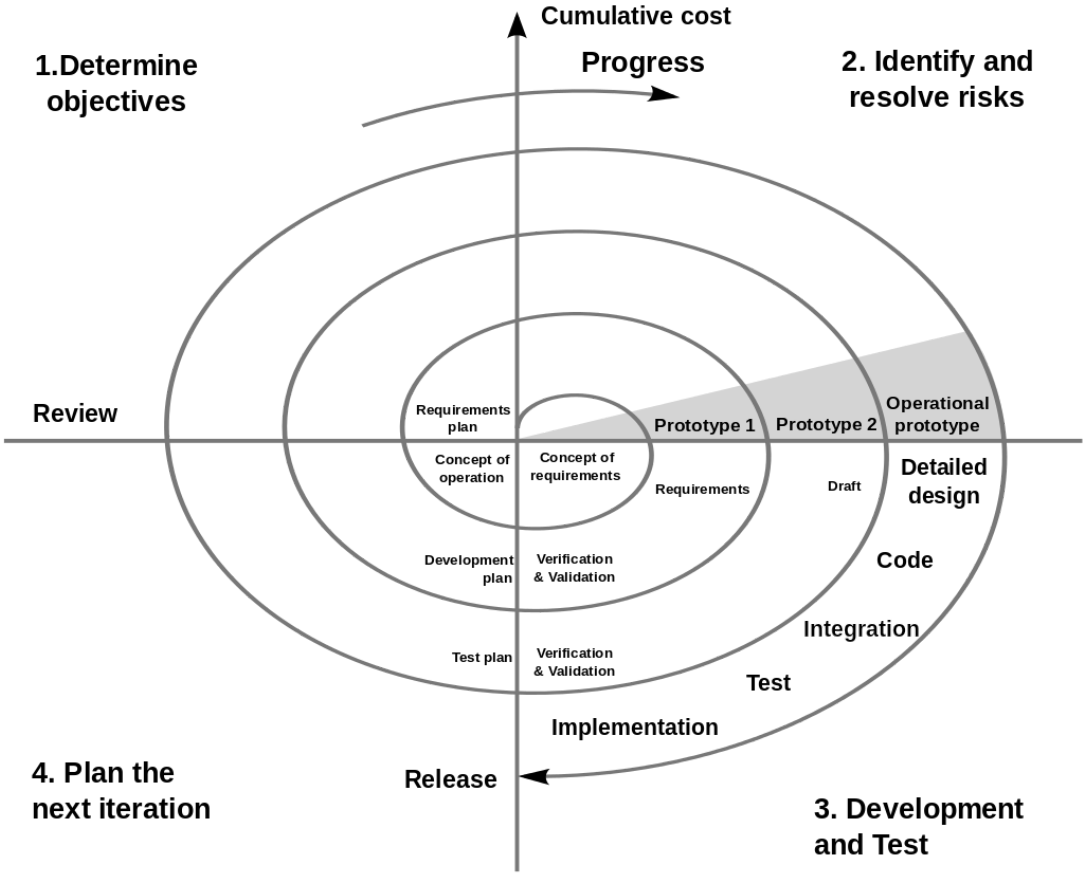
- **Satisfy the Customer -** Early and continuous delivery of valuable software.
- **Welcome Change -** Harness change for a competitive advantage.
- **Deliver Frequently -** Maintaining a constant pace for completed work.
- **Work Together -** Customers are part of the team.
- **Build Projects -** Providing motivated individuals with the environment and support they need and trusting them to get the job done.
- **Face-To-Face Time -** Assembling the project team and business owners on a daily basis throughout the project.
- **Measure of Progress -** Measuring progress by the amount of completed work.
- **Sustainable Development -** Creating processes that promote sustainable efforts.
- **Continuous Attention -** Continuous attention to technical excellence and good design enhances agility.
- **Keep It Simple -** Breaking big work down into smaller tasks that can be completed quickly.
- **Organized Teams -** The best architectures, requirements, and designs emerge from self-organizing teams.
- **Reflect for Effectiveness -** Having the team reflect at regular intervals on how to become more effective, then tuning and adjusting behavior accordingly.

SCRUM

Scrum is a subset of Agile. It is a lightweight process framework for agile development, and the most widely-used one. A Scrum process is distinguished from other agile processes by specific concepts and practices, divided into the three categories of Roles, Artifacts, and Time Boxes.
A “process framework” is a particular set of practices that must be followed in order for a process to be consistent with the framework. For example, the Scrum process framework requires the use of development cycles called Sprints.
“Lightweight” means that the overhead of the process is kept as small as possible, to maximize the amount of productive time available for getting useful work done.

Spiral Methods

Spiral Model is a combination of a waterfall model and iterative model. Each phase in the spiral model begins with a design goal and ends with the client reviewing the progress. The spiral model was first mentioned by Barry Boehm in his 1986 paper.
The development team using the Spiral-SDLC model starts with a small set of requirements and goes through each development phase for those set of requirements. The software engineering team adds functionality for the additional requirement in every-increasing spirals until the application is ready for the production phase.



Each phase of Spiral Model is divided into four quadrants as shown in the image above. The functions of these four quadrants are:

1. **Determine Objectives:** Requirements are gathered from the customers and the objectives are identified, elaborated and analyzed at the start of every phase. Then alternative solutions possible for the phase are proposed in this quadrant.
2. **Identify and Resolve Risks:** During the second quadrant all the possible solutions are evaluated to select the best possible solution. Then the risks associated with that solution is identified and the risks are resolved using the best possible strategy. At the end of this quadrant, prototype is built for the best possible solution.
3. **Development and Test:** During the third quadrant, the identified features are developed and verified through testing. At the end of the third quadrant, the next version of the software is available.
4. **Plan the Next Iteration:** In the fourth quadrant, the customers evaluate the so far developed version of the software. In the end, planning for the next phase is started.

Computational Platforms

In order to use methods such as agile development there have been tools developed. This allows them to be more efficient. The tools have various purposes including collaboration, code and automation on a server. The development may also be done on a cloud to allow for faster and concurrent access, this allows the continuous feedback agile requires.

Collaboration

Collaboration is a requirement when working in a team. Using services such as Slack and Trello, allow more efficient use of time. They allow instant communication and assignment of tasks. These tasks may be made as small or large as required. This allows the easy subdivision amongst the team members, as well as a constant gauge of the progress being made. These services also allow chats to be created for specific areas of the projects and/or tasks, which makes it easy for the collection of information in a single source.

Code

Software Engineering is mostly about the product, and most of the product is often comprised mostly of code. Therefore the management of this code is paramount. There are services such as Github, Bitbucket, and Bamboo, as well as tools such as Git, which allow this management to become easier, as well as creating failsafes and back ups. These services create backups of the code on a cloud, however, they are much more involved than this. There is the ability to create branches, which means that teams can work on their own parts of the same code separately, without affecting anyone elses then merge it back with the main block of code once it has been checked whether the new piece of code is compatible or not.

Automation on a Server

Most services are now run off of servers instead of people local machines, this means that there have been many tools built exclusively to make the process of deploying to a server then running and maintaining that server much easier and more efficient. These services include Amazon Web Services (AWS), Docker, Microsoft Windows Azure and Google Compute Engine. Many of these services sell servers as a service which allows the expensive and difficult job that is maintaining servers and having redundancies to be made much easier, since many of these services give an all inclusive package, therefore this means that the service can be deployed using, often given tools, then maintenance is minor.

Algorithmic Approaches

There are a few approaches available to measure and assess the measure the software engineering process. However, the most effective method used is machine learning, since it allows the process itself to make connections and assessments on those connections that may not have been thought of otherwise.

Machine Learning

Machine learning is the scientific study of algorithms and statistical models that computer systems use to perform a specific task without using explicit instructions, relying on patterns and inference instead. It is seen as a subset of artificial intelligence. Machine learning algorithms build a mathematical model based on sample data, known as "training data", in order to make predictions or decisions without being explicitly programmed to perform the task.

There are various types of machine learning algorithms:

- **Supervised Learning -** The machine learning task of learning a function that maps an input to an output based on example input-output pairs. It infers a function from labeled training data consisting of a set of training examples. Supervised learning algorithms include:
    - **Classification** - Used when the outputs are restricted to a limited set of values.
    - **Regression** - Used when the outputs may have any numerical value within a range.
- **Unsupervised Learning** - A type of self-organized Hebbian learning that helps find previously unknown patterns in data set without pre-existing labels. It is also known as self-organization and allows modeling probability densities of given inputs.
- **Reinforcement Learning** - An area of machine learning concerned with how software agents ought to take actions in an environment so as to maximize some notion of cumulative reward. Reinforcement learning algorithms do not assume knowledge of an exact mathematical model of the Markov Decision Process, and are used when exact models are infeasible.
    - A Markov decision process (MDP) is a discrete time stochastic control process. It provides a mathematical framework for modeling decision-making in situations where outcomes are partly random and partly under the control of a decision maker.

Ethical Concerns

Ethics or moral philosophy is a branch of philosophy that involves systematizing, defending, and recommending concepts of right and wrong conduct.
There is a software engineering code of ethics. Software engineers shall commit themselves to making the analysis, specification, design, development, testing and maintenance of software a beneficial and respected profession. In accordance with their commitment to the health, safety and welfare of the public, software engineers shall adhere to the following Eight Principles:

- **Public** – Software engineers shall act consistently with the public interest.
- **Client and Employer** – Software engineers shall act in a manner that is in the best interests of their client and employer consistent with the public interest.
- **Product** – Software engineers shall ensure that their products and related modifications meet the highest professional standards possible.
- **Judgment** – Software engineers shall maintain integrity and independence in their professional judgment.

- **Management** – Software engineering managers and leaders shall subscribe to and promote an ethical approach to the management of software development and maintenance.
- **Profession** – Software engineers shall advance the integrity and reputation of the profession consistent with the public interest.
- **Colleagues** – Software engineers shall be fair to and supportive of their colleagues.
- **Self** – Software engineers shall participate in lifelong learning regarding the practice of their profession and shall promote an ethical approach to the practice of the profession.

The question then is, is tracking the progress and performance of teams, then acting upon that data to improve the current strategy or change the strategy entirely ethical? The answer, in my opinion, is yes and no. This is due to the fact that in theory it is a great idea and should absolutely be implemented, however, in reality the world is more complex, mostly due to, in my opinion two reasons. There may be an invasion of privacy connected with tracking of all of this data. People expect certain freedoms, and rightly so. Therefore the amount of data that can be tracked is limited in this regard. The second thought is that it may put unnecessary pressure on people to perform. This may be a good short term solution, however, in the long run this may cause unhealthy levels of stress which may lead to poor health, physical and mental.

Therefore, the true question becomes, how does one balance tracking data and improving performance with the humans that are meant to perform.

Conclusion

This report discusses the ways in which the software engineering process can be measured and assessed in terms of measurable data, different methods to make the process of software development more efficient, the various computational platforms that may be used to improve productivity and adherence to the development methods, algorithmic approaches to the development process and finally the ethical considerations of developing software.

References

https://www.techopedia.com/definition/13296/software-engineering - Software Engineering Definition
http://wiki.c2.com/?EngineeringPrinciples - Engineering Principles
https://en.wikipedia.org/wiki/History_of_software_engineering - History of Software Engineering
https://www.sebokwiki.org/wiki/Software_Engineering_Features_-_Models,_Methods,_Tools,_Standards,_and_Metrics - Metrics
https://www.cprime.com/resources/what-is-agile-what-is-scrum/ - Defining Agile
https://www.360logica.com/blog/the-importance-of-different-agile-methodologies-included-in-agile-manifesto/ - Agile Image
https://www.cgi.com/us/en-us/life-sciences/blog/12-principles-of-agile-methodologies - Agile principles
https://searchcio.techtarget.com/definition/Agile-Manifesto - Agile principles
https://en.wikipedia.org/wiki/Spiral_model#/media/File:Spiral_model_(Boehm,_1988).svg - Spiral Model image
https://www.guru99.com/what-is-spiral-model-when-to-use-advantages-disadvantages.html - Defining the Spiral Model
https://www.researchgate.net/publication/311497124_A_Framework_for_Agile_Development_in_Cloud_Computing_Environment - Computational Methods
https://en.wikipedia.org/wiki/Machine_learning - Defining Machine Learning
https://en.wikipedia.org/wiki/Markov_decision_process - Defining Markov Decision Process
https://www.computer.org/education/code-of-ethics - Code of Ethics