Measuring Software Engineering

Table of Contents

Requirement

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.

Introduction

"Software engineering is an engineering discipline that is concerned with all aspects of software production from the early stages of system specification to maintaining the system after it has gone into use. In this definition, there are two key phrases:

1. *Engineering discipline* Engineers make things work. They apply theories, methods and tools where these are appropriate. Engineers also recognize that they must work to organizational and financial constraints.
2. *All aspects of software production* Software engineering is not just concerned with the technical processes of software development but also with activities such as software project management and with the development of tools, methods and theories to support software production."

– (Sommerville, 1982)

Definition of Software Engineering

To understand how and why we measure aspects of software engineering we must first have a firm grasp of the field. Software engineering encompasses the design, development, testing, deployment, maintenance and management of software systems.
The study of software engineering came about as an answer to the issues of low-quality software. Problems arise in software development most commonly when a development cycle exceeds its budget, timeline and produces subpar quality of code. Software engineering aims to ensure software is built to the client's requirements, consistently coded, on time and within budget. Software engineering also emerged to tackle the problems of rapidly changing user requirements and the ever-changing environments in which the code must run.

Fields of Software Engineering

Software Engineering is a discipline of engineering which itself contains many subfields cover the overall lifecycle of software development. The list below gives a comprehensive overview of the different aspects of software engineering as set out in SWEBOK (Software Engineering Body of Knowledge)

- Software requirements (or Requirements engineering): The elicitation, analysis, specification, and validation of requirements for software.
- Software design: The process of defining the architecture, components, interfaces, and other characteristics of a system or component. It is also defined as the result of that process.
- Software construction: The detailed creation of working, meaningful software through a combination of programming (aka coding), verification, unit testing, integration testing, and debugging
- Software testing: An empirical, technical investigation conducted to provide stakeholders with information about the quality of the product or service under test.

- Software maintenance: The totality of activities required to provide cost-effective support to software.
- Software configuration management: The identification of the configuration of a system at distinct points in time for the purpose of systematically controlling changes to the configuration, and maintaining the integrity and traceability of the configuration throughout the system life cycle. Modern processes use software versioning.
- Software engineering management: The application of management activities—planning, coordinating, measuring, monitoring, controlling, and reporting—to ensure that the development and maintenance of software is systematic, disciplined, and quantified.
- Software development process: The definition, implementation, assessment, measurement, management, change, and improvement of the software life cycle process itself.
- Software engineering models and methods: imposing structure on software engineering with the goal of making that activity systematic, repeatable, and ultimately more success-oriented
- Software quality
- Software engineering professional practice: is concerned with the knowledge, skills, and attitudes that software engineers must possess to practice software engineering in a professional, responsible, and ethical manner
- Software engineering economics: is about making decisions related to software engineering in a business context
- Computing foundations
- Mathematical foundations
- Engineering foundations - (IEEE, 2014)

Measurable Data

There are many was of trying to measure the process of software engineering. However, none of them are decidedly the best method or combination of methods for measuring engineer performance or overall quality of the software development process.
For example, one very basic way of measuring the development process is LOC (lines of code). But this method can be ambiguous and has many issues. First off there are different ways of counting LOC some systems will count each new line, but many developers disagree as this figure can include lines of dead code, spacing lines and comments.
To circumvent this issue LOC can be measured by counting only the logical operations. This still causes problems in development however. For example, if a development team puts a lot of

emphasis on volume of code and errors, then software developers may avoid dealing with complicated problems instead focusing on writing lots of simple code to keep their LOC up and their error count down.

There are plenty of metrics easily capturable in the software development process,

- LOC
- Error count
- Commits
- Keystrokes
- Active days

While this data can give great insights into the development process, it is important to remember that these metrics could prove completely useless and could even be detrimental to the development process, i.e. causing software developers to change their behaviour to satisfy an arbitrary system of performance measurement.

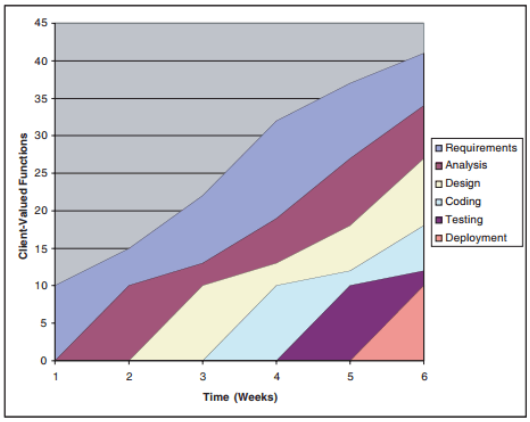
Agile Production Metrics

The actual writing of code is not the only part of the development process that can be empirically measured. A team using agile development process will use metrics that help inform project decision and planning, these don't describe the software, but they are invaluable when it comes to improving the development process.

Lead Time

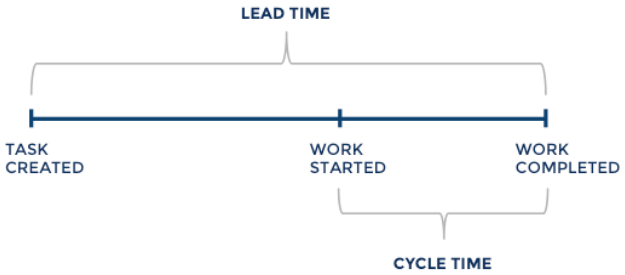
This is a metric to measure the time taken for ideas to go from inception to deployment. A lower lead time means the team is very responsive to client requirements and changes

Lead Time



Cycle Time

Cycle time is a measure of the elapsed time when work starts on an item (story, task, bug etc.) until it's ready for delivery. Cycle time tells how long (in calendar time) it takes to complete a task. This metric is a subsection of lead time above. Being able to provide an idea of when features will be completed helps to manage expectations and avoid unpleasant surprises.



Production Metrics

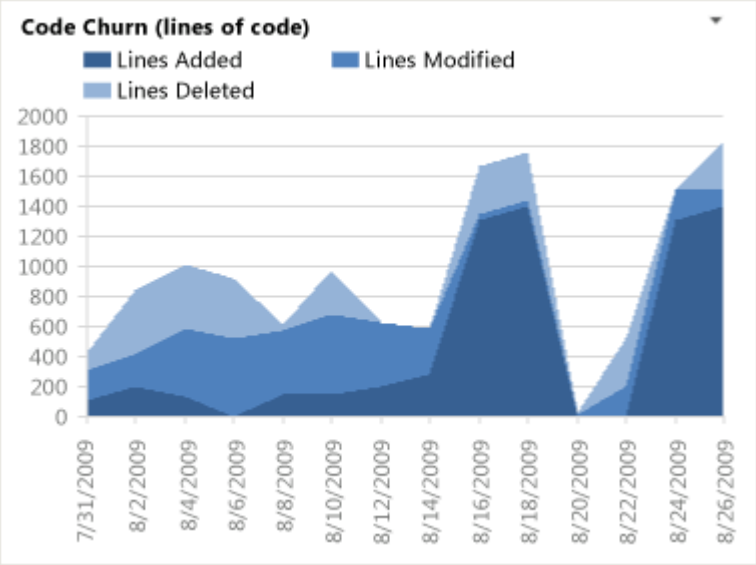
These metrics attempt to measure how much work is done and determine the efficiency of software development.

Active Days

As mentioned before active days is the amount of time a developer spends writing code for the team. Active days is an important measure because it can help to build up an idea of how long certain tasks will take. Active days does not include any administration time and as such it can also allow you to gauge how much development time interruptions such as meetings have on development.

Code Churn

This is a measure of the number of line s of code that were modified, added or deleted from the code base over a specified period. If The code churn on a task increases dramatically it may indicate that the task needs more attention. Usual over the development lifetime of a task the level of code churn will fluctuate (e.g. high code churn during the initial exploratory phase and steadily decreasing leading up to release).

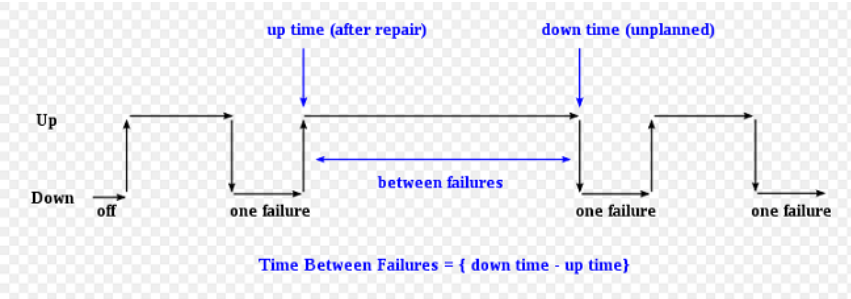


Impact Metrics

Impact metrics aim to measure the effect of code changes on the software. For example, a refactoring of a large portion of the code will likely have many more repercussions on the development process as opposed to an addition of an entirely new file/feature or just a small refactoring and/or update to the code.

MTBF and MTTR

Mean time between failures and mean time to recover both measure the software's performance in a production environment. Since software failures are inevitable, these metrics attempt to quantify how well the software recovers from such an incident and how quickly it can be brought back online.



Application Crash Rate (ACR)

This is a simple measurement to keep track of how often the software crashes it's calculated by dividing the application fails by how many times it is used.

$$\boldsymbol{ACR = \frac{Fails}{Uses}}$$

Other Metric Categories

- Size Oriented Metrics
  - Errors per Kilo LOC
  - Defects per Kilo LOC
  - Cost per Kilo LOC
- Function oriented metrics
  - Focuses on the functionality a software package delivers.
- Security Metrics
  - These measures reflect the software security quality.
  - Also includes time to respond metrics in the event of a breach

Methods of Gathering and Analysing Data

Above I have outlined an array of software development metrics that can be empirically measured to give insight into the development process. Collecting these metrics is another matter entirely. For this information to be useful to a development team it needs accurate and up to date metrics on its own performance. Unfortunately, this can be quite laborious and time consuming in terms of logging huge amounts of development data and there is the issue that developers may unfaithfully or inaccurately record their own performance data. In addition to this if these metric collection practices are not in place from the beginning of the development process, they can be a nightmare to adopt depending on the methods employed for collecting the data. I will now explore some solutions to these issues.

Computational Platforms

There are several tools available for gathering information on the software development process. For almost three decades the Collaborative Software Development Laboratory at the University of Hawaii has conducted research and development in software engineering practices. They have been continuously developing software platforms to help development teams track development metrics and improve the overall development process and the quality of the software they produce.

Personal Software Process (PSP)

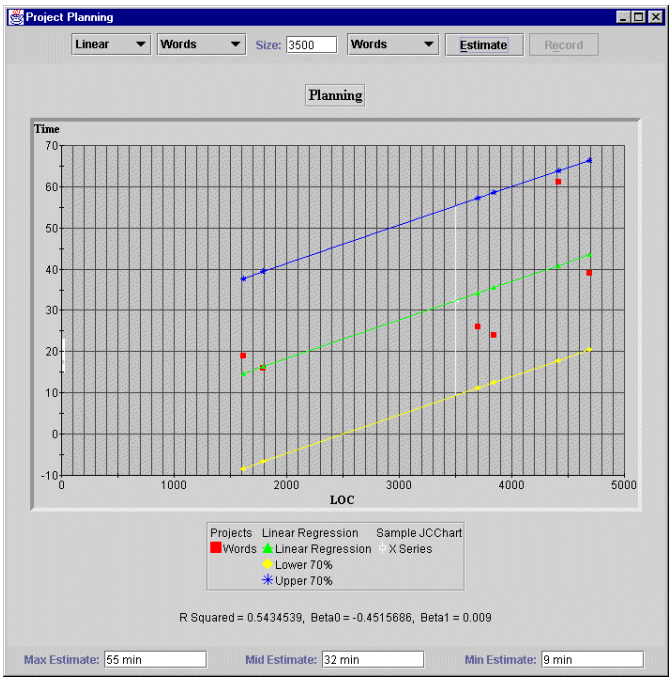
One of the first attempts they made to quantify software development metrics, this platform was designed to help dev teams:

- Improve their estimating and planning skills.
- Make commitments they can keep.
- Manage the quality of their projects.
- Reduce the number of defects in their work

While this new system had many benefits. It was largely manual in nature requiring developers to fill in forms and complete checklists about tasks they completed and what went well and what went wrong. This whole process took a lot of additional effort on the part of the developers. Because of the need for all the data to ne inputted manually a lot of human error was introduced and many concerns were raised as to the quality of the data collected and fears it would lead the development teams planning in the wrong direction.

Leap

In response to the problems with the PSP system, the LEAP toolkit was developed. LEAP stands for **L**ightweight, **E**mpirical, **A**utomated, and **P**ortable Software Developer Improvement, It's a bit of a mouthful. LEAP aimed to circumvent the data quality concerns by automating the data analysis. However, this system did still require a lot of the data to be entered by the developers.



Hacky Stat

Hackystat is an open source framework for collection, analysis, visualization, interpretation, annotation, and dissemination of software development process and product data. Hackystat users typically attach software 'sensors' to their development tools, which unobtrusively collect and send "raw" data about development to a web service called the Hackystat "SensorBase" for storage. The "SensorBase" repository can be queried by other web services to form higher level abstractions of this raw data, and/or integrate it with other internet-based communication or coordination mechanisms, and/or generate visualizations of the raw data, abstractions, or annotations. (Johnson, 2001)

What was so enticing about Hacky Stat was the automation with for both PSP and LEAP the developers were required to enter the data themselves. This new system was met with a whole new set of ethical concerns and push back, mainly from the developers being asked to use it. While Hackystat claims to be "unobtrusive" it collected a broad amount of data that could be made easily available to managers.

Ethical Issues

Monitoring and storing information on employee performance have become common place in all industries and businesses. No matter the job however the reasons for keeping track of such information remains the same, companies collect data on the workers to gain valuable insights into the productivity of their employees and processes. Data mining and analysis has become a huge market in recent times and while it is most noticeable in terms of marketing and advertisement it has had arguably a greater effect within companies and how they manage their employees. The information companies collect on their employees can boost productivity and efficiency immensely. However, where should we draw the line? Where does the data collection cross over from purely work related to more personal information?

In my opinion data collection and analysis to improve productivity and efficiency is a fantastic thing and should be practiced more frequently and, in all occupations/industries to produce the greatest outcome for everyone involved. At the same time, I recognise that very often these practices can be intrusive, and the data collected misused or sold to the highest bidder. I have no problem when the data collected on an employee as long as that data relates solely to the role the employee occupies within the company.

To further my point on collecting only relevant data I would like to argue that constantly monitoring every action of employees while they're at work would have a detrimental effect on the work force. When put under constant surveillance people will change their behaviour and will never have a chance to properly relax. You may argue that people should not be relaxed at work and should be constantly focused on the task at hand this is an unhealthy attitude and it's an unmaintainable standard to keep. Inevitably such pressure on employees will lead to them making more mistakes or becoming burnt out.

Conclusion

The collection of data has had an overall beneficial impact on the software development process. It has led to better development to better development processes, increasing productivity and reducing the number of errors and defects in our software products. However, with the ever-increasing pace of software developments and integration of new technologies we must be careful, no more than ever, about how we implement these new technologies that could monitor us both in the work place and at home. The ethical concerns of these technologies will require constant discussion and debate if we are to keep up with their rapid development.

References

IEEE, 2014. *www.swebok.org.* [Online]
Available at: https://www.computer.org/web/swebok/v3
[Accessed 24 May 2016].
Johnson, P., 2001. *HackyStat.* [Online]
Available at: http://csdl.ics.hawaii.edu/research/hackystat/
[Accessed 19 Nov 2018].
Sommerville, I., 1982. *Software Engineering.* 8th ed. Harlow, England: Pearson Education.