Measuring Software Engineering

Jason Mullen - 18320585

Table of Contents

Introduction

This report aims to provide a brief overview on measuring software engineering, the metrics, platforms and algorithms used for measuring, as well as some of the advantages and ethical concerns that come with it.

Reasons For Measuring Software Engineering

Measuring software engineering (or using software metrics) can be utilised most importantly to predict key factors of software such as its reliability, efficiency, functionality, required resources and much more. Knowing these aspects can help in allowing software to be successfully implemented as it was intended to be. In a world in which our reliance on software is increasing every year, it is essential that implementing software into the real world can be accurately planned ahead of time, for the sake of maximizing the use of resources etc, with the confidence that it will run as intended.

History

In order to discuss the measuring of software engineering in both present and future times, it is important to reflect on the history of software engineering metrics, where they succeeded and where they failed.
The first attempt to predict software quality by using software metrics was in 1971[1] when Akiyama proposed a model for calculating the number of errors per KLOC (Thousands of Lines of Code). However it became apparent that using this single measurement in order to determine the many different aspects of software quality was not adequate. Software engineers then began working on creating metrics for these different aspects. The 1970's saw large amounts of progress in measuring software complexity and size. The benefits of software metrics became more widely recognized and soon there were company-wide software metric programs being organized.[2] However even in the 1990's, despite the huge interest in software metrics, most companies failed to utilize the modern innovations in measuring software:

" The industrial take-up of most academic software metrics work has been woeful. Much of the increased metrics activity in industry is based almost entirely on metrics that were around in the early 1970s "[3]

Nowadays though with most businesses relying heavily on software, metrics have become increasingly more popular and necessary.

[1] Fenton, N. E., and Martin, N. (1999) "Software metrics: successes, failures and new directions." Journal of Systems and Software 47.2 pp. 149-157.

[2] Fenton, N. E., and Martin, N. (1999) "Software metrics: successes, failures and new directions." Journal of Systems and Software 47.2 pp. 149-157.

[3] Fenton, N. E., and Martin, N. (1999) "Software metrics: successes, failures and new directions." Journal of Systems and Software 47.2 pp. 149-157.

Metrics Used To Measure Software Engineering[4]

Software metrics have become increasingly more precise throughout the years, as when developing software there are many many diverse characteristics which need to be considered and measured using different metrics. These metrics can be broken down into different categories and are as follows:

- **Management Metrics**
  - Size: Lines of Code (LOC*), Thousand Lines of Code (KLOC)
  - 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 – Defects Removed/Hour (for review and test)
  - Test Coverage
  - Failure Rate
- **Software Requirements Metrics**
  - Change requests (received, open, and 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

[4]https://www.sebokwiki.org/wiki/Software_Engineering_Features_-_Models,_Methods,_Tools,_Standards,_and_Metrics

- **Software Maintenance and Operation**
    - Mean Time Between Changes (MTBC)
    - Mean Time to Change (MTTC)
    - System Reliability
    - System Availability
    - Total Hours of Downtime

Platforms For Measuring

Many platforms have been developed for measuring software with each having their own advantages and disadvantages.

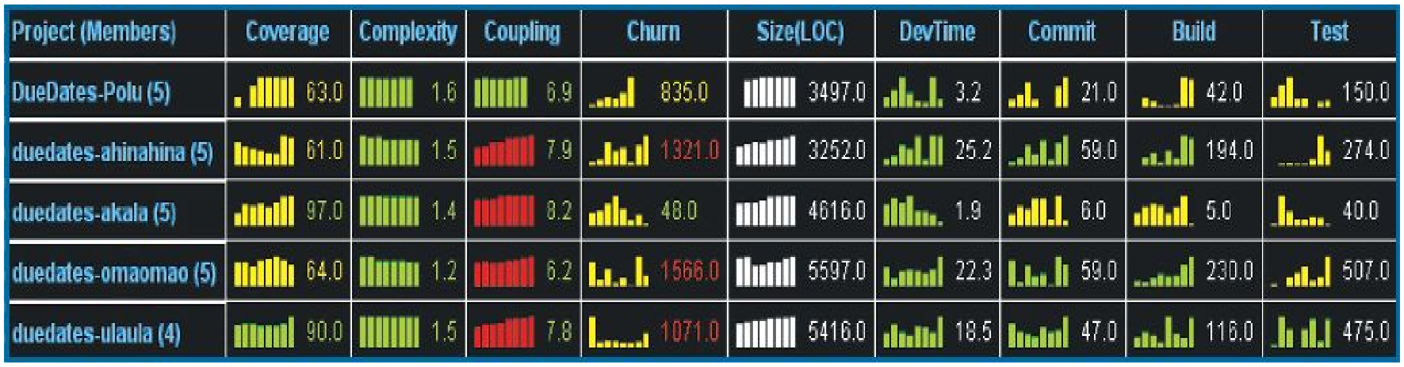
- **The Personal Software Process (PSP)[5]**

The PSP provides a way to track a developer’s performance and progress throughout the development life of a product. By analyzing development metrics such as number of bugs discovered and development time, PSP can help determine the future of the development of the product, and highlight areas that the developer can improve in.

- **Hackystat[6]**

Hackystat is an example of a PSP which has ceased development but can still be accessed on github. This particular tool collects the development metrics through sensors attached to development tools that communicate with a server using the Simple Object Access (SOAP) protocol. It is extremely secure in terms of privacy and focuses on the individual. It also uses XML files to store data so that there is no need for a back-end database.

[5] A. Sillitti, A. Janes, G. Succi, and T. Vernazza, “Collecting, integrating and analyzing software metrics and personal software process data,” in Proceedings of the 29th Euromicro Conference. IEEE
[6] A. Sillitti, A. Janes, G. Succi, and T. Vernazza, “Collecting, integrating and analyzing software metrics and personal software process data,” in Proceedings of the 29th Euromicro Conference. IEEE



Project (Members)	Coverage	Complexity	Coupling	Churn	Size(LOC)	DevTime	Commit	Build	Test
DueDates-Polu (5)	63.0	1.6	6.9	835.0	3497.0	3.2	21.0	42.0	150.0
duedates-ahinahina (5)	61.0	1.5	7.9	1321.0	3252.0	25.2	59.0	194.0	274.0
duedates-akala (5)	97.0	1.4	8.2	48.0	4616.0	1.9	6.0	5.0	40.0
duedates-omaomao (5)	64.0	1.2	6.2	1566.0	5597.0	22.3	59.0	230.0	507.0
duedates-ulaula (4)	90.0	1.5	7.8	1071.0	5416.0	18.5	47.0	116.0	475.0

Figure 1. Example of Hackystat information

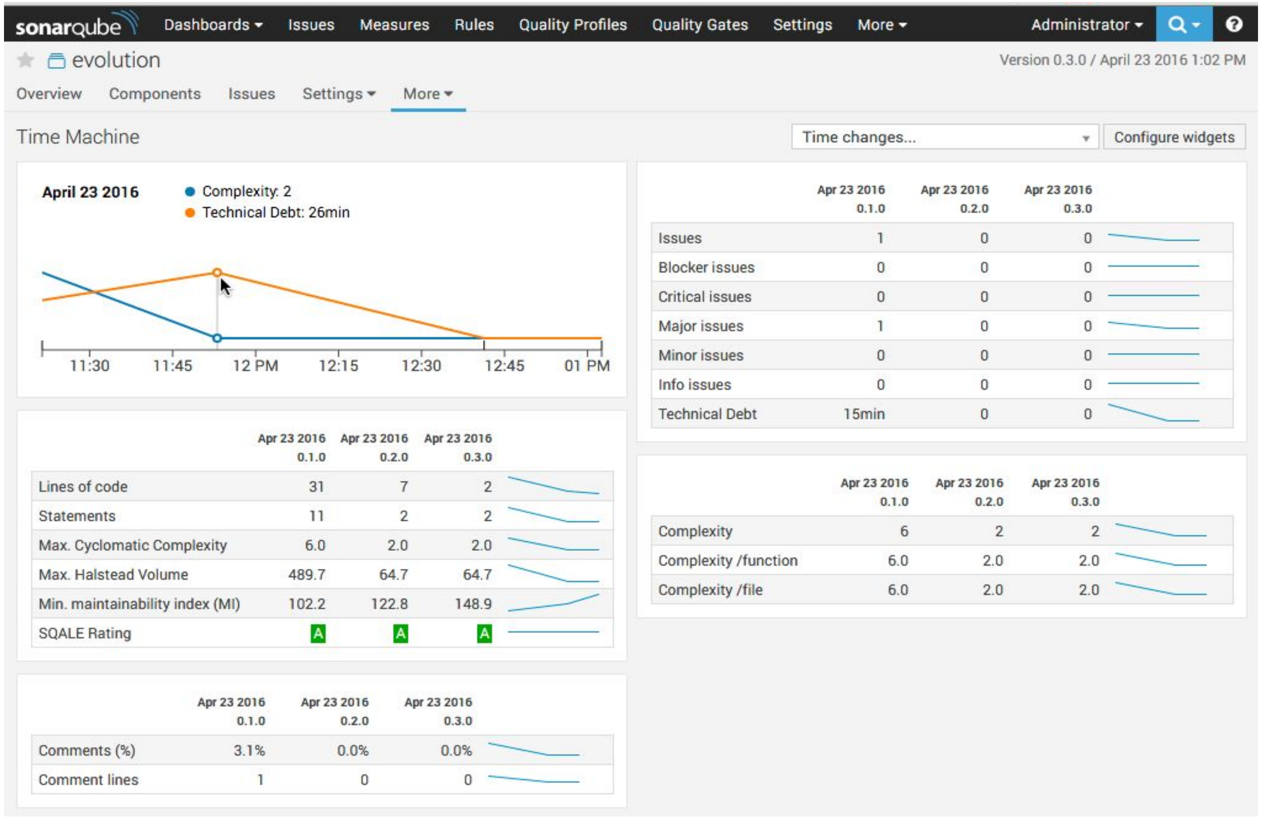
- **SonarQube**

SonarQube is a platform for measuring software which offers support for[7] Java, C#, PHP, JavaScript, TypeScript, C/C++, Ruby, Kotlin, Go, COBOL, PL/SQL, PL/I, ABAP, VB.NET, VB6, Python, RPG, Flex, Objective-C, Swift, CSS, HTML, and XML. SonarQube aims to cover quality on seven aspects of software:[8]

- Duplicated Code
- Coding Standards
- Unit Tests
- Complex Code
- Potential Bugs
- Commenting
- Design and Architecture

SonarQube offers a unique tool called TimeMachine which allows the user to look back on the history of Sonar's reporting on these seven aspects and view their evolution. This is not only useful for predicting the future development process based on the past, but also for determining the effectiveness of measuring software engineering.

[7] https://www.sonarqube.org/features/multi-languages/
[8] http://www.methodsandtools.com/PDF/mt201001.pdf



	Apr 23 2016 0.1.0	Apr 23 2016 0.2.0	Apr 23 2016 0.3.0
Issues	1	0	0
Blocker issues	0	0	0
Critical issues	0	0	0
Major issues	1	0	0
Minor issues	0	0	0
Info issues	0	0	0
Technical Debt	15min	0	0

	Apr 23 2016 0.1.0	Apr 23 2016 0.2.0	Apr 23 2016 0.3.0
Lines of code	31	7	2
Statements	11	2	2
Max. Cyclomatic Complexity	6.0	2.0	2.0
Max. Halstead Volume	489.7	64.7	64.7
Min. maintainability index (MI)	102.2	122.8	148.9
SQALE Rating	A	A	A

	Apr 23 2016 0.1.0	Apr 23 2016 0.2.0	Apr 23 2016 0.3.0
Complexity	6	2	2
Complexity /function	6.0	2.0	2.0
Complexity /file	6.0	2.0	2.0

	Apr 23 2016 0.1.0	Apr 23 2016 0.2.0	Apr 23 2016 0.3.0
Comments (%)	3.1%	0.0%	0.0%
Comment lines	1	0	0

Figure 2. SonarQube's TimeMachine

Algorithmic Approaches

Computer scientists love applying algorithms to everything and anything and measuring the effectiveness of their own algorithms is no different. The benefits are obvious as when done right, algorithms can measure software much more efficiently and without the defects of human errors.

Halstead's Software Metrics[9]

This automated approach involves declaring six values and using various formulas which include these values to determine software metrics such as program length, volume, effort. In addition to the six values, some metrics use other metric results in their equations also. For example.

[9] https://www.geeksforgeeks.org/software-engineering-halsteads-software-metrics/

First we define the initial six values

- n1 = Number of distinct operators.
- n2 = Number of distinct operands.
- N1 = Total number of occurrences of operators.
- N2 = Total number of occurrences of operands.
- n1* = Number of potential operators.
- n2* = Number of potential operands.

The equation for Program Volume is

- V = Size * (log2 vocabulary) = N * log2(n)

And the equation for Potential Minimum Volume is

- V* = (2 + n2*) * log2(2 + n2*)

Then in order to determine the Program Level, the results are taken from the two previously mentioned equations and used in the equation for Program Level

- L = V* / V

The Cyclomatic Number[10]

Cyclomatic complexity[11] is an algorithm that is used to determine the complexity of a program based on the number of linearly independent paths in it. Using the flow graph of a program, the cyclomatic number is calculated and its formula is as follows

V(G) = E - N + P.
Where

- G is the program's flow graph
- E is the number of edges
- N is the number of vertices

[10] http://cs.ndsu.edu/~perrizo/saturday/papers/paper/CATA-2003%20Papers/206.pdf
[11] https://www.softwareyoga.com/cyclomatic-complexity/

- P is the number of connected components

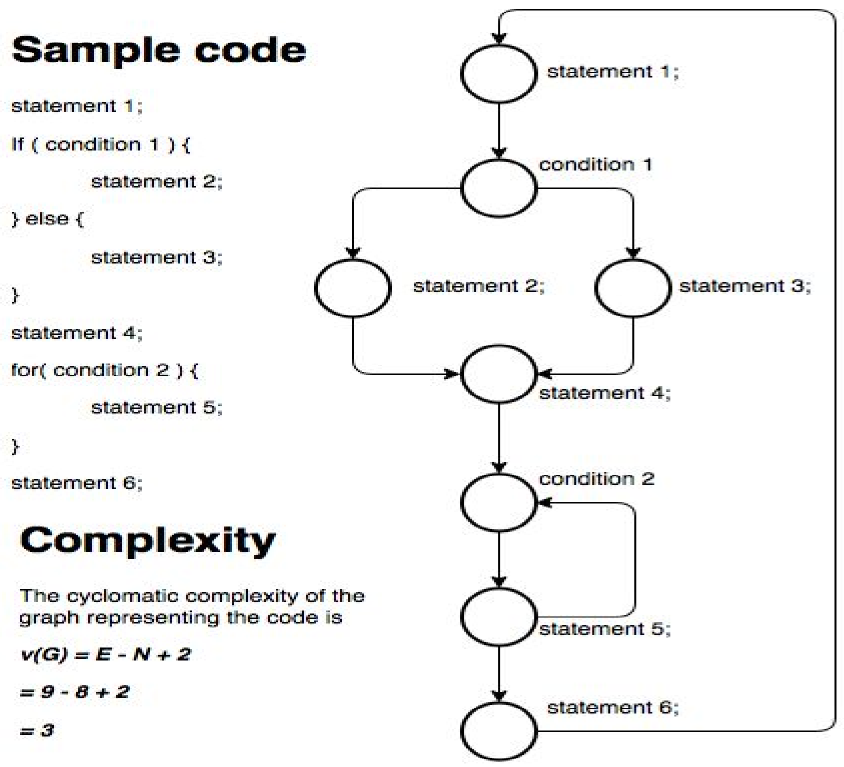


Figure 3. Example of a program's flow graph and cyclomatic number calculation[12]

This result of the cyclomatic number is equal to the number of independent cycles in graph G.

Duane Growth Model[13]

This is a reliability growth model which determines the reliability of a system as a function of time. In other words it is used to measure the change in reliability of a program. Duane's model achieves this by plotting the cumulative failure rate against the cumulative hours on log-log paper using the formulas

[12] https://www.softwareyoga.com/cyclomatic-complexity/
[13] https://www.osti.gov/servlets/purl/10105800

$\lambda c(t) = N(t) / t = at^{-\beta}$

$\log(\lambda c) = \log(a) - \beta\log(t)$

Where

- $N(t)$ = cumulative number of failure
- t = total time
- λc = cumulative failure rate
- a, β = parameters

Ethics

Most ethical concerns come from the effect that using software metrics has on the software developers. I have discussed the advantages of using software metrics and will now highlight some of the possible ethical concerns that come with them

Measurement of The Individual

I don't think that it would be a stretch to say that the majority of software developers would prefer their work to not be measured, and possibly compared against their other peers. Having the knowledge that your work efficiency, effectiveness, readability etc. is constantly being tracked could produce the opposite effect intended and distract the developer rather than aid them. Not to mention the stress that it could induce on developers, especially on ones working in already stressful circumstances such as crunch time[14] leading up to an important release. In the ideal world, businesses would only use these software measures to help improve a developer's skills and make them a more efficient member of a team. However, since we do not live in an ideal world, it is certainly plausible that some businesses may use these measures as a threat/incentive to prevent workers from slacking. The use of these measures could also create a competitive environment in which programmers withhold information from their coworkers and impede their progress in order to appear more favourable when

[14] https://www.cbr.com/cd-projekt-red-cyberpunk-2077-crunch-time-problem/

their statistics are compared. And in the inevitable scenario where programmer's statistics get compared, it is important to question if task difficulties are taken into account when measures are being assessed. For example a developer working on a completely new mechanic for a software will take much longer and most likely write more complex, less streamline, illegible code when compared to a developer bringing an old, already coded functionality from previous software to new. There is also a probability that developers will build their programs around the metrics used to measure quality instead of focusing on the actual purpose of the software they are programming.

Privacy of Measurements

Another ethical concern to consider is who is able to view the results. Can coworkers see each other's statistics? Possibly resulting in a 'blame-game' or alienation for developers who's measurements lag behind the rest. Most importantly, are there any measures in place to prevent businesses from sharing their worker's data with other businesses. This could create a world in which companies already have a profile of a programmer prior to any interview. And hiring new employees is as simple as comparing the job candidate's statistics to each other and selecting the most favourable option as if creating a fantasy football team. Needless to say there are many other factors that need to be considered when hiring a programmer such as how they work in teams, if they live relatively near to the workplace etc. However it is also not a farfetched fantasy to imagine a scenario in which workers are employed based on a profile of their statistical history coming into fruition.

Conclusion

It is apparent that, when used for pure purposes (i.e to genuinely improve the quality of a developer) there aren't too many ethical concerns that need to be considered when measuring software engineering. However when it is used to the detriment of the developer and hung over their heads, it can have many resulting knock-on effects that in most scenarios will hinder the software production.