Measuring Software Engineering
CS3012 - Software Engineering
Emily Harte
17330411

SOFTWARE MEASUREMENT

The measurement of software is known as software metrics. A metric is measure of characteristics of software or the software engineering process that are quantifiable or countable.[1] There have been many methods developed over the years for the measure of software and software complexity as software engineering as emerged as a discipline. Henry and Kaufra authored an article in the IEEE Transactions on Software Engineering journal in 1981. They developed a metric for software complexity known as Software Structure Metrics Based on Information Flow. This method sees complexity as a function of fan-in and fan-out. Fan-in was defined as a procedure as the number of local flows into that procedure plus the number of data structures from which that procedure retrieves information. The authors defined fan-out as the number of local flows out of that procedure plus the number of data structures that the procedure updates.[2]

Maurice Howard Halstead was a professor at Purdue university and the originator of software science.[3] He introduced his method of software metrics, Halstead Complexity Measures in 1977. According to Halstead's method, metrics of software should mirror the implementation or expression of algorithms in different languages. However, he also put forward that the metrics should be independent of their execution on a specific platform.[4]

Some general methods of software measurement include Cohesion and Coupling. Cohesion involves the degree to which the elements inside a software module belong together. It quantifies the depth of the relationship between the methods and data of a class in the code. It also measures the unifying function served by that class.

---

[1] Stringfellow, A., 2017, *What are Software Metrics and How Can You Track Them?*, DZone, accessed 20th October 2019, <https://dzone.com/articles/what-are-software-metrics-and-how-can-you-track-th>

[2] Henry, S., Kafura, 1981, I*EEE Transactions on Software Engineering Volume SE-7*, Issue 5, pp. 510 - 518

[3] Lee, J.A.N., 2017, *Computer Pioneers*, IEEE Computer Society, accessed 20th October 2019, <https://history.computer.org/pioneers/halstead.html>

[4] Halstead, M. H. (1977). *Elements of Software Science*. Amsterdam: Elsevier North-Holland, Inc.

Furthermore, it quantifies the strength of relationship between the class' methods and the data within.[5] With regards to Object-Oriented Programming, if the methods that serve a class tend to be complementary, then the class is said to have high cohesion. Code readability and reusability is increased in a highly cohesive system. Complexity is also kept manageable.[6]

Coupling is the extent of interdependence between software modules. It also assesses how closely related two routines or modules are.[7] It also determines the strength of relationship between modules.[8] Coupling is juxtaposed by cohesion. Low coupling often corresponds to high cohesion, and vice versa.[9] Coupling and Cohesion were conceived by Larry Constantine, an American software engineer, based on the attributes of "good" programming practices that achieved minimum maintenance and modification costs.[10]

However, there are many limitations to measuring and quantifying software. Binstock argues that to fully define or measure software qualities is challenging. He also considers that the determination of a valid and concurrent measurement metric is laborious. Determining which metrics matter and their meaning is elusive.[11] Kaner asserts that some software development professionals indicate that simplistic measurements can do more harm than good.[12] However, Binstock further claims that metrics have become an integral part of the software development process.[13] Lincke, Lundberg and Löwe

---

5 Yourdon, E., Constantine, L. L. (1979). *Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design*. Yourdon Press.

6 Marsic I. (2012). *Software Engineering.* Rutgers University

7 ISO/IEC/IEEE 24765, 2010, *Systems and software engineering — Vocabulary*

8 ISO/IEC TR 19759, 2005, *Software Engineering — Guide to the Software Engineering Body of Knowledge (SWEBOK)*

9 Kaye, D. (2003). *Loosely Coupled: The Missing Piece of Web Services*. Rds Pr.

10 Stevens, W. P., Myers, G. J., Constantine, L. L. (1974). "Structured design". *IBM Systems* Journal, vol 13, no. 2, pp. 115–139.

11 Binstock, A., 2010, *Integration Watch: Using metrics effectively,* SD Times, BZ Media, Accessed 22nd October 2019, <https://sdtimes.com/metrics/integration-watch-using-metrics-effectively/>

12 Kaner, C., Walter, B., 2004, 'Software Engineer Metrics: What do they measure and how do we know?', 10th International Software Metrics Symposium

13 Lincke, R., Lundberg, J, Löwe, W., (2008), *Comparing software metrics tools*, Växjö University

disagree, arguing that the interpretation of many methodologies are imprecise and the tools for computing them are ambiguous as to how they arrive at a particular result.[14]

Lines of code, the amount of lines of code in a program, is a common software metric that has the ability to measure or predict the amount of time and effort that will be required to develop a program. It is also a useful tool to measure programming productivity and maintainability. It is debated how to properly measure lines of code. However, orders of magnitude and scaling of lines of code is a useful comparator. This can be an indicator of software complexity.[15]

```
for (int i= 0; i < 100; i++) printf("%d\n", i);
```

Lines of code measures are generally broken down into two categories: physical lines of code and logical lines of code. Regarding the above line of code, we would consider there to be one physical line of code and two logic lines of code (one for the for loop and one for the printf statement). The amount of physical lines of code changes depending on the coding standards of a particular institution or software engineer. This is another considerable factor when attempting to measure the size of a program. Comment lines are generally included when measuring the amount of lines of code in a program.[16]

Software productivity is the measure of ability of a development team to build and maintain software systems. There is no single definition for productivity in software engineering, which constitutes a hindrance to the discussion of and setting a standard for software productivity.[17] However, there is common agreement that productivity describes the ratio between manufacturing input and manufacturing output.[18] This generally true in a manufacturing context. In the case of software engineering, where the main capital is not material goods but the worker's knowledge, factors beside input and output must be

---

[14] Ibid

[15] Nguyen, V., Deeds-Rubin, S. Tan, T., Boehm, B., (2007), *A SLOC Counting Standard*, Center for Systems and Software Engineering, University of Southern California

[16] Ibid

[17] Neal, A., Hesketh, B., Anderson, N., Ones, D. S., Sinangil, H. K., Viswesvaran, C. (2002). *Handbook of Industrial, Work and Organizational Psychology Productivity in Organizations*. Sage Publications Ltd. p.8-24.

[18] Chew, B. W., 1998, "No-Nonsense Guide to Measuring Productivity", *Harvard Business Review*, no. 66, pp. 110-115

taken into account when measuring productivity. These include quality of work, timeliness and project success.[19]

Software maintainability is a key issue when developing and measuring software. Maintainability involves the dexterity with which the program can be modified after release to correct faults, bugs or defects and to improve performance.[20] The categories of maintenance include adaptive, adapting the system to changes in software environment, and corrective, diagnosing and fixing errors.[21] Factors such as spaghetti code, unstructured code cause by a lack of programming style standards, make software difficult to maintain.[22]

Computational Platforms

There are various platforms available to perform measurements on software. A challenge in today's industry is to determine the most effective programs and platforms to effectively measure software engineering. A further challenge is to integrate them into existing work practices. New methods may be resisted by over-worked and short staffed software development teams. [23]

Sharma and Kaulgud created a tool called *PIVoT* that collects and analyses of in-project and project health metrics.[24] Guosios and Spinellis released a tool focuses on open-source project repositories before the widespread adoption of Github. It uses conventional version control and issue tracking systems.[25] Their currently available tool,

---

[19] Drucker, P. F., 1999, "Knowledge-Worker Productivity: The Biggest Challenge", *California Management Review*, no. 41, pp. 79-94

[20] ISO/IEC 14764, 2006, *Software Engineering — Software Life Cycle Processes — Maintenance*

[21] *Software Maintenance and Re-engineering*, CSE2305 Object-Oriented Software Engineering

[22] Markus, P., 2004, "Straightening spaghetti-code with refactoring?", *Software Engineering Research and Practice*, pp. 846–852

[23] Thiruvathukal, G.K., Hayward, N. J., Laufer, K., 2018, "*Metrics Dashboard: A Hosted Platform for Software Quality Metrics*", Loyola University Chicago

[24] Sharma, V.S., Kaulgud, V., 2012, "PIVoT: Project insights and Visualization Toolkit," *3rd International Workshop on Emerging Trends in Software Metrics (WETSoM)*, pp. 63–69.

[25] Gousios, G., Spinellis, D., 2009, "A platform for software engineering research," *6th IEEE International Working Conference on Mining Software Repositories (MSR)*, pp. 31–40

GHTorrent, mines Github commits from hundreds of thousands of repositories. The data is available for "deep crawling" in a public MongoDB database.[26]

Thiruvathukal et al developed the *Metric Dashboard*, a platform for tracking metrics by mining open-source repositories on Github. It determines the metrics characteristic of team progress. The platform allows the user to submit a URL of a repository for analysis. The user can then study various metrics over time visually. Supported metrics on this platform include project size, issue spoilage and productivity.[27]

Ben Thompson and Travis Kimmel founded GitPrime in 2015.[28] It is an organisational tool providing engineers with software metrics to increase visibility about team contributions and identify areas to give feedback.[29] GitLab is a similar tool that provides a Git-repository manager. It has features such as wiki and issue-tracking.[30]

Version Control is the management of changes to documents, code files and programs. Version control systems are usually stand-alone systems. Version control is critical in software engineering for teams to track the progress of the development and to correct potential mistakes in the code.[31] Github is a prolific host for version control using Git, it provides features such as bug tracking and task management.[32]

Ericsson, Löwe et al developed an architecture called *VizzAnalyzer* to combine program analysis and software visualisation techniques. They aimed to answer research questions such as "does a synergy of program analysis and visualisation really leverage software understanding" and "what is a reasonable tradeoff between precision of the analyses and scalability in manageable legacy systems size". They aimed to apply the tool onto itself to look for design flaws uncovered by the analyses.[33]

---

[26] 2012, "GHTorrent: Github's data from a firehose", *9th IEEE Working Conference on Mining Software Repositories (MSR)*, pp. 12–21.

[27] Thiruvathukal, G.K., Hayward, N. J., Laufer, K., 2018, "*Metrics Dashboard: A Hosted Platform for Software Quality Metrics*", Loyola University Chicago

[28] GitPrime, crunchbase, accessed 25th October 2019, < https://www.crunchbase.com/organization/gitprime#section-overview>

[29] Jaala, 2019, *What is GitPrime exaclty?*, GitPrime, accessed 25th October 2019<https://help.gitprime.com/general/what-is-gitprime-exactly>

[30] Gitlab, accessed 25th October 2019, <https://about.gitlab.com/features/gitlab-ci-cd/>

[31] O'Sullivan, B. (2009). *Mercurial: the Definitive Guide*. Sebastopol: O'Reilly Media, Inc.

[32] Ibid

[33] Lowe, W. et al., "VizzAnalyzer– A Software Comprehension Framework", University of Vaxjo

There are various metrics tools based in the Eclipse IDE. Analyst4j is available as a stand-alone application or as an Eclipse plug-in. It visualises code quality using automated software metrics and graphs.[34] Eclipse Metrics Plug-in 1.3.6. was created by Frank Sauer as an open-source dependency analyser for Eclipse. It can detect cycles, packages, and type dependencies.[35] Semmle, another Eclipse plug-in, provides an SQL-like query language for object-oriented code. This enables the search of bugs.[36]

Research has shown that the use of social tools to track software engineering productivity within organisations is increasing. This is in part due to the broad adoption of cloud and mobile platforms to deploy these technologies.[37] A social technology is a way of using human, intellectual and digital resources in order to influence social process, which is vital to the team-based software engineering process.[38]

According to the McKinsey Global Survey results, the number of companies that see large benefits from use of social technologies is growing. However, half of organisations say that the benefit is minimal within their companies. There appear to be many benefits from utilising social technology within an organisation. Social tools contribute 20 percent of revenue increases and 18 percent of cost improvements in a company. These increases are even larger when tools are on a mobile device. 65% of companies report that they use at least one social tool on mobile. Despite these advantages, there are some risks associated with social technologies. There is a constant worry of leaks of confidential information. Also, large financial gains are more likely at smaller companies, leaving larger organisations disillusioned.[39]

---

[34] VRP Consulting, accessed 25th October 2019, <http://www.codeswat.com>

[35] sauerf, 2013, *Eclipse Metric plugin*, SourceForge, accessed 25th October 2019, <http://sourceforge.net/projects/metrics>

[36] Semmle, accessed 25th October 2019, <http://semmle.com>

[37] Bughin, J., Chui, M., 2012, "Evolution of the networked enterprise: McKinsey Global Survey results", McKinsey

[38] Tamošiūnaitė, R.(2018). *Socialinių technologijų taikymo galimybės gyventojų dalyvavimui viešojo valdymo sprendimų priėmimo procesuose*. Vilnius: Mykolas Romeris University. p. 11.

[39] Bughin, J., Chui, M., 2012, "Evolution of the networked enterprise: McKinsey Global Survey results", McKinsey

Algorithmic Approaches

The complexity of a problem is the amount of resources required for an exhaustive solution to the problem. Algorithms are inherent in computer programming and indeed, software engineering.[40] A measure of the software science effort of a program *P* can be obtained by using the estimate of the program level *L(P)* and program volume *V(P).*

Hence, we get the formula:

E(P) = V(P) / L(P)

where **E** is the *effort* of the program.[41]

The cyclomatic number V(P) of a program P when the complexity measure is based on a control flow graph model is given by the formula:

V(P) = e – n + 2

where e is the number of edges and n is the number of nodes
in the control flow graph of a program P[42]

This algorithm is known as cyclomatic complexity. It was developed by Thomas J. McCabe, Sr. in 1976. He showed that the cyclocmatic complexity of a given program with only one entry point and one exit point is equal to the number of decision points in the program plus one.[43] This algorithm can be used to limit complexity of routines during program development. Per McCabes recommendation, the software engineer should count the complexity of the modules they are developing and split them up accordingly into smaller modules where the cyclomatic complexity of the module is greater than 10.[44]

---

[40] Debnath, N., Burgin, M., “Software Metrics from the Algorithmic Perspective”, Winona State University, University of California, Los Angeles

[41] Halstead, M. H. (1977). *Elements of Software Science*. Amsterdam: Elsevier North-Holland, Inc.

[42] McCabe, T. J., 1976, “A Complexity Measure”, *IEEE Transaction on Software Engineering*, pp. 308- 320

[43] Fricker, S., 2018, *What Exactly is Cyclomatic Complexity?”*, froglogic, accessed 26th October 2019, <https://www.froglogic.com/blog/tip-of-the-week/what-is-cyclomatic-complexity/>

[44] McCabe, T. J., 1976, “A Complexity Measure”, *IEEE Transaction on Software Engineering*, pp. 308- 320

When creating Halstead complexity measures, Halstead's goal was to identify measurable parts of software, and the relations between them. For a given program, let:

$\mathbf{n_1}$ = number of distinct operators

$\mathbf{n_2}$ = number of distinct operands

$\mathbf{N_1}$ = total number of operators

$\mathbf{N_2}$ = total number of operands

Several metrics can be calculated from these numbers:

Program vocabulary: $n = n_1 + n_2$

Program length: $N = N_1 + N_2$

Volume: $V = N \times \log_2 n$

Difficulty: $D = n_1 / 2 \times N_2 / n_2$

Effort: $E = D \times V$

The effort measure translates into real world time to code the program using the following:

Time required to program: $T = E / 18$ [45]

A genetic algorithm is procedure used to generate solutions to optimisation in a program. It used biology inspired operators such as mutation and selection.[46] Affinity Propagation is a clustering algorithm that uses the concept of message passing between data points.[47] It can be used for computational biology efforts. It can also be used for data mining, such as text data mining. Yang et al proposed a strategy combing these these two concepts to analyse software metrics and predict software quality. The strategy

[45] Halstead, M. H. (1977). *Elements of Software Science*. Amsterdam: Elsevier North-Holland, Inc.

[46] Mitchell, M., (1996). *An Introduction to Genetic Algorithms*. Cambridge: MIT Press. p. 2

[47] Frey, B. J., Dueck, D., 2007, "Clustering by Passing Messages between Data Points", *Science*, Vol. 315, no. 5814, pp. 972 - 976

has been validated using two software metric datasets. It was shown that it performs well in software metrics.[48]

Ethical Concerns

One of the most important concerns in terms of ethics in Software Engineering is to protect programmers and individual engineers from harm by ensuring anonymity when measuring the software engineering process. When conducting the empirical study of Software Engineering, the confidentiality of the engineer's defect data is vital. However, when analysing metrics on an institutional scale, the deficiencies in a programmer's work may be known to the company. Another concern is that the protection of employees can conflict with the interests of the company.[49] When conducting outside research, consent is an important issue. The consent of an organisation is required when obtaining access to their source code or documents.[50] It is debatable whether the consent of the creators of the data, i.e. programmers, is also required when consent from the larger institution is granted. Both the US and Canadian guidelines for conducting research in the area of Software Engineering does not require the consent of organisations or programmers as individuals when the information is in the public domain.

On an institutional level, when measuring the Software Engineering process in the form of metrics, programmers and engineers do not expect their code to remain private within the company. Generally, code written for company projects is the property of the company. As a result, the need to obtain the consent of programmers is effectively eliminated. When analysing software metrics, the company will already have found defects in the source code. Following on from this, the company can determine the number of flaws introduced by each individual programmer. When contracted to complete work for a company, programmers can expect their work to be reviewed by the organisation as laid out in their contract. The individual is not placed at any additional risk

---

48 Yang, B., Chen, X., Xu, S., Guo, P., (2008). "Software Metrics Analysis with Genetic Algorithm and Affinity Propagation Clustering". *Proceedings of the 2008 International Conference on Data* Mining, pp. 590-596.

49 Singer , J., Vinson, N. G., 2002, "Ethical Issues in Empirical Studies of Software Engineering", *IEEE Transactions On Software Engineering*, vol. 28, pp. 1171 - 1180

50 Jeffries, D.R., Votta, L., 1999, "Empirical Software Engineering: Guest Editor's Special Section Introduction," *IEEE Transactions on Software Engineering*, Vol. 25, No. 4, pp. 435-437

to their employment by metrics obtained by the company. Therefore, it is not necessary to obtain their consent from an ethical perspective.[51]

In regards to research conducted in the field of Software Engineering, there are some exceptions to the obligation of attaining informed consent. If the data is already anonymised and no individual can be identified from it, informed consent and confidentially are generally not required. Another exception arises when more harm to the individual or organisation emerges from preserving confidentiality than from breaching it. Where permitted by law, confidentiality can be breached to protect an individual from harm.[52]

Software Engineers influence the final product when participating in a software development process. This can include actions that may be contrary to public interest. Ethical behaviour in the Software Engineering process is considered dynamic, rather than static. Using a code of ethics, ethical behaviour of software engineers can be regulated and taught. The Software Engineering Code of Ethics delivered by the ACM and IEEE Computer Society is one such convention. It outlines that the engineering team should treat the maintenance of programs with the same professionalism as the development of programs. It advocates that the 'public interest' is central to the code in terms of the health, safety and welfare of the public. It demands that software engineers ensure that their final products meet the highest professional standards possible.[53]

In terms of confidentiality, the Code of Ethics requires the software practitioner to maintain the privacy of any confidential information obtained in their professional work, whether it relates to the company or the public. Its caveat, however, is that the confidentiality is consistent with public interest and the law. The engineer is expected to use only accurate data obtained by ethical and lawful means, as well as to use it in approved ways. It also asks the engineer to maintain the integrity of data.

When measuring the process of Software Engineering on a company level, the Code of Ethics asserts that it is the engineer's duty to identify if a project is likely to fail and inform their client or employer immediately. The engineer should document and collect evidence that will attest to this potential failure. The same procedure is required if

---

51 Singer , J., Vinson, N. G., 2002, "Ethical Issues in Empirical Studies of Software Engineering", *IEEE Transactions On Software Engineering*, vol. 28, pp. 1171 - 1180

52 Anderson, R., 1992"Social Impacts of Computing: Codes of Professional Ethics," *Social Science Computing Review*, Vol. 10, no. 2, pp. 453-469

53 Gotterbarn D., Miller, K. W., 2010, "Unmasking Your Software's Ethical Risks", *IEEE Software*, Vol. 27, no. 1, pp. 12 -13

the project will run over its budget, will violate intellectual property law or be otherwise questionable. It is the engineer's, as well as the company's, responsibility to provide realistic measures of cost. When analysing metrics, the engineer or company should not minimise the estimates of scheduling, personnel or quality of a project.[54]

[54] Ibid

Bibliography

2012, “GHTorrent: Github’s data from a firehose”, 9th IEEE Working Conference on Mining Software Repositories (MSR), pp. 12–21.

Anderson, R., 1992“Social Impacts of Computing: Codes of Professional Ethics,” Social Science Computing Review, Vol. 10, no. 2, pp. 453-469

Bughin, J., Chui, M., 2012, "Evolution of the networked enterprise: McKinsey Global Survey results”, McKinsey

Bughin, J., Chui, M., 2012, "Evolution of the networked enterprise: McKinsey Global Survey results”, McKinsey

Debnath, N., Burgin, M., “Software Metrics from the Algorithmic Perspective”, Winona State University, University of California, Los Angeles

Frey, B. J., Dueck, D., 2007, “Clustering by Passing Messages between Data Points”, Science, Vol. 315, no. 5814, pp. 972 - 976

Fricker, S., 2018, What Exactly is Cyclomatic Complexity?”, froglogic, accessed 26th October 2019, <https://www.froglogic.com/blog/tip-of-the-week/what-is-cyclomatic-complexity/>

Gitlab, accessed 25th October 2019, <https://about.gitlab.com/features/gitlab-ci-cd/>

GitPrime, crunchbase, accessed 25th October 2019, < https://www.crunchbase.com/organization/gitprime#section-overview>

Gotterbarn D., Miller, K. W., 2010, “Unmasking Your Software’s Ethical Risks”, IEEE Software, Vol. 27, no. 1, pp. 12 -13

Gousios, G., Spinellis, D., 2009, “A platform for software engineering research,” 6th IEEE International Working Conference on Mining Software Repositories (MSR), pp. 31–40

Jaala, 2019, What is GitPrime exaclty?, GitPrime, accessed 25th October 2019<https://help.gitprime.com/general/what-is-gitprime-exactly>

Jeffries, D.R., Votta, L., 1999, “Empirical Software Engineering: Guest Editor’s Special Section Introduction,” IEEE Transactions on Software Engineering, Vol. 25, No. 4, pp. 435-437

Lowe, W. et al., “VizzAnalyzer– A Software Comprehension Framework”, University of Vaxjo

Markus, P., 2004, "Straightening spaghetti-code with refactoring?”, Software Engineering Research and Practice, pp. 846–852

Mitchell, M., (1996). An Introduction to Genetic Algorithms. Cambridge: MIT Press. p. 2

O'Sullivan, B. (2009). Mercurial: the Definitive Guide. Sebastopol: O'Reilly Media, Inc.

sauerf, 2013, Eclipse Metric plugin, SourceForge, accessed 25th October 2019, <http://sourceforge.net/projects/metrics>

Semmle, accessed 25th October 2019, <http://semmle.com>

Singer , J., Vinson, N. G., 2002, “Ethical Issues in Empirical Studies of Software Engineering”, IEEE Transactions On Software Engineering, vol. 28, pp. 1171 - 1180

Tamošiūnaitė, R.(2018). Socialinių technologijų taikymo galimybės gyventojų dalyvavimui viešojo valdymo sprendimų priėmimo procesuose. Vilnius: Mykolas Romeris University. p. 11.

Thiruvathukal, G.K., Hayward, N. J., Laufer, K., 2018, “Metrics Dashboard: A Hosted Platform for Software Quality Metrics”, Loyola University Chicago

VRP Consulting, accessed 25th October 2019, <http://www.codeswat.com>

Yang, B., Chen, X., Xu, S., Guo, P., (2008). “Software Metrics Analysis with Genetic Algorithm and Affinity Propagation Clustering”. Proceedings of the 2008 International Conference on Data Mining, pp. 590-596.

Binstock, A., 2010, Integration Watch: Using metrics effectively, SD Times, BZ Media, Accessed 22nd October 2019, from https://sdtimes.com/metrics/integration-watch-using-metrics-effectively/

Chew, B. W., 1998, “No-Nonsense Guide to Measuring Productivity”, Harvard Business Review, no. 66, pp. 110-115

Drucker, P. F., 1999, “Knowledge-Worker Productivity: The Biggest Challenge”, California Management Review, no. 41, pp. 79-94

Halstead, M. H. (1977). Elements of Software Science. Amsterdam: Elsevier North-Holland, Inc.

Henry, S., Kafura, 1981, IEEE Transactions on Software Engineering Volume SE-7, Issue 5, pp. 510 - 518

ISO/IEC 14764, 2006, Software Engineering — Software Life Cycle Processes — Maintenance

ISO/IEC TR 19759, 2005, Software Engineering — Guide to the Software Engineering Body of Knowledge (SWEBOK)

Kaner, C., Walter, B., 2004, ‘Software Engineer Metrics: What do they measure and how do we know?’, 10th International Software Metrics Symposium

Kaye, D. (2003). Loosely Coupled: The Missing Piece of Web Services. Rds Pr.

Lee, J.A.N., 2017, Computer Pioneers, IEEE Computer Society, accessed 20th October 2019, from https://history.computer.org/pioneers/halstead.html

Lincke, R., Lundberg, J, Löwe, W., (2008), Comparing software metrics tools, Växjö University

Marsic I. (2012). Software Engineering. Rutgers University

Neal, A., Hesketh, B., Anderson, N., Ones, D. S., Sinangil, H. K., Viswesvaran, C. (2002). Handbook of Industrial, Work and Organizational Psychology Productivity in Organizations. Sage Publications Ltd. p. 8-24.

Nguyen, V., Deeds-Rubin, S. Tan, T., Boehm, B., (2007), A SLOC Counting Standard, Center for Systems and Software Engineering, University of Southern California

Sharma, V.S., Kaulgud, V., 2012, “PIVoT: Project insights and Visualization Toolkit,” 3rd International Workshop on Emerging Trends in Software Metrics (WETSoM), pp. 63–69.

SO/IEC/IEEE 24765, 2010, Systems and software engineering — Vocabulary

Software Maintenance and Re-engineering, CSE2305 Object-Oriented Software Engineering

Stevens, W. P., Myers, G. J., Constantine, L. L. (1974). "Structured design". IBM Systems Journal, vol 13, no. 2, pp. 115–139.

Stringfellow, A., 2017, What are Software Metrics and How Can You Track Them?, DZone, accessed 20th October 2019, from https://dzone.com/articles/what-are-software-metrics-and-how-can-you-track-th

Yourdon, E., Constantine, L. L. (1979). Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design. Yourdon Press.