

Maintainability Index

YEGOR BUGAYENKO

Lecture #5 out of 24

80 minutes

The slidedeck was presented by the author in this [YouTube Video](#)

All visual and text materials presented in this slidedeck are either originally made by the author or taken from public Internet sources, such as web sites. Copyright belongs to their respected authors.





FRED BROOKS

“The total cost of maintaining a widely used program is typically 40 percent or more of the cost of developing it”

— Fred Brooks. *The Mythical Man-Month: Essays on Software Engineering*. Addison-Wesley Publishing Company, 1982. doi:[10.1002/spe.4380060417](https://doi.org/10.1002/spe.4380060417)



WARD CUNNINGHAM

“Shipping first time code is like going into debt. A little debt speeds development so long as it is paid back promptly with a rewrite. The danger occurs when the debt is not repaid. Every minute spent on not-quite-right code counts as interest on that debt.”

— Ward Cunningham. Experience Report — The WyCash Portfolio Management System. In *Proceedings of the Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)*, pages 29–30, 1992. doi:[10.1145/157710.157715](https://doi.org/10.1145/157710.157715)



PAUL OMAN

“Before developers can claim that they are building maintainable systems, there must be some way to measure maintainability”

— Don Coleman, Dan Ash, Bruce Lowther, and Paul Oman. Using Metrics to Evaluate Software System Maintainability. *Computer*, 27(8), 1994.
[doi:10.1109/2.303623](https://doi.org/10.1109/2.303623)



“The factors of software that determine or influence maintainability can be organized into a hierarchical structure of measurable attribute. Our hierarchy serves as a taxonomic definition for software maintainability that is compatible with the 35 published works upon which it is based.”

— Paul Oman and Jack Hagemeister. Metrics for Assessing a Software System’s Maintainability. In *Proceedings of the International Conference on Software Maintenance*, pages 337–338. IEEE, 1992. doi:[10.1109/icsm.1992.242525](https://doi.org/10.1109/icsm.1992.242525)

Software Maintainability Taxonomy



Source: Paul Oman and Jack Hagemester. Metrics for Assessing a Software System's Maintainability. In *Proceedings of the International Conference on Software Maintenance*, pages 337–338. IEEE, 1992.
doi:[10.1109/icism.1992.242525](https://doi.org/10.1109/icism.1992.242525)

Maintainability Formula

To quantify the maintainability of a tree we can then use the following formula:

$$\prod_{i=1}^m W_{D_i} \left(\frac{\sum_{j=1}^n W_{A_j} M_{A_j}}{n} \right)_i$$

where

W_{D_i} = Weight of influence of maintainability
Dimension D_i

W_{A_j} = Weight of influence of maintainability
Attribute A_j

M_{A_j} = Metric or measure of maintainability
Attribute A_j

“This formula represents the product of the weighted dimensions, where each dimension is measured as the average deviation from a known value of ‘goodness’ for that maintainability attribute.”

Source: Paul Oman and Jack Hagemeister. Metrics for Assessing a Software System’s Maintainability. In *Proceedings of the International Conference on Software Maintenance*, pages 337–338. IEEE, 1992.
[doi:10.1109/icsm.1992.242525](https://doi.org/10.1109/icsm.1992.242525)



“A software maintainability model is only useful if it can provide developers and maintainers in an industrial setting with more information about the system”

— Don Coleman, Dan Ash, Bruce Lowther, and Paul Oman. Using Metrics to Evaluate Software System Maintainability. *Computer*, 27(8), 1994. doi:10.1109/2.303623

First Approximation

$$\begin{aligned} \text{Maintainability} &= 171 \\ &- 3.42 \times \ln(\text{ave}E) \\ &- 0.23 \times \text{ave}V(g') \\ &- 16.2 \times \ln(\text{ave}LOC) + \text{ave}CM \end{aligned}$$

where *aveE*, *aveV(g')*, *aveLOC*, and *aveCM* are the average effort, extended V(G), average lines of code, and number of comments per submodule (function or procedure) in the software system.

“Approximately 50 regression models were constructed in an attempt to identify simple models that could be calculated from existing tools and still be generic enough to apply to a wide range of software systems. The regression model that seemed most applicable was a four-metric polynomial based on 1) Halstead’s effort, 2) extended cyclomatic complexity, 3) lines of code, and 4) number of comments.”

The Formula of Maintainability Index

$$\begin{aligned} \text{Maintainability} = & 171 \\ & - 5.2 \times \ln(\text{aveVol}) \\ & - 0.23 \times \text{ave } V(g') \\ & - 16.2 \times \ln(\text{aveLOC}) \\ & + (50 \times \sin(\sqrt{2.46 \times \text{perCM}})) \end{aligned}$$

aveVol — average Halstead Volume in a module

ave V(g') — average total cyclomatic complexity in a module

aveLOC — average lines of code in a module

perCM — average percent of comments in a module

Maintainability Index by Visual Studio

$$MI = \max \left[0, 100 \frac{171 - 5.2 \ln V - 0.23G - 16.2 \ln L}{171} \right]$$

Source: Introduction to Code Metrics, by Radon

	MI >= 20	High Maintainability
	10 <= MI < 20	Moderate Maintainability
	MI < 10	Low Maintainability

Source: Think Twice Before Using the “Maintainability Index”, by Arie van Deursen

“We decided to be conservative with the thresholds. The desire was that if the index showed red then we would be saying with a high degree of confidence that there was an issue with the code.” — Code metrics — Maintainability index range and meaning by Microsoft, 2011.



RAINER NIEDERMAYR

“We are convinced that Maintainability Index is nonsense. We think that it is not sensible to reduce the maintainability of a whole software system to one single indicator.”

— Rainer Niedermayr. Why We Don't Use the Software Maintainability Index. <https://teamscale.com/blog/en/news/blog/maintainability-index>, 2016. [Online; accessed 15-03-2024]

“The Maintainability Index does not provide information about the impact on development activities. A value of 57 does not express which maintainability aspects are affected by a bad value.” — Rainer Niedermayr



ARIE VAN DEURSEN

“If you are a researcher, think twice before using the maintainability index in your experiments. Make sure you study and fully understand the original papers published about it.”

— Arie van Deursen. Think Twice Before Using the “Maintainability Index”.
<https://avandeursen.com/2014/08/29/think-twice-before-using-the-maintainability-index/>, 2014. [Online; accessed 15-03-2024]



“Tool smiths and vendors used the exact same formula and coefficients as the 1994 experiments, without any recalibration.” — Arie van Deursen



TIM GILBOY

“If we’re going to use the Maintainability Index we should use it to measure relative maintainability within our project rather than use it as an absolute metric.”

— Tim Gilboy. Maintainability Index — What Is It and Where Does It Fall Short? <https://sourcery.ai/blog/maintainability-index/>, 2022. [Online; accessed 15-03-2024]



“Extending the length can significantly decrease Maintainability Index, even if all of the changes cause the code to be clearer and more understandable.” — Tim Gilboy



TJAŠA HERIČKO

“When comparing maintainability measurements from several Index variants, the perception of maintainability could be impacted by the choice of the Index variant used.”

— Tjaša Heričko. Exploring Maintainability Index Variants for Software Maintainability Measurement in Object-Oriented Systems. *Applied Sciences*, 2023. doi:[10.3390/app13052972](https://doi.org/10.3390/app13052972)

Maintainability Index is supported by a few tools:

- Visual Studio for C++ and others
- SonarQube for Java
- Testwell for Java and C++
- Radon for Python
- jscomplexity for JavaScript
- maintidx for Go

Hindawi

Scientific Programming

Volume 2021, Article ID 5840399, 25 pages

<https://doi.org/10.1155/2021/5840399>

Hindawi

Review Article

A Tool-Based Perspective on Software Code Maintainability Metrics: A Systematic Literature Review

Luca Ardito¹,^{*} Riccardo Coppola,¹ Luca Barbato,² and Diego Verga¹

¹*Politecnico di Torino Department of Control and Computer Engineering Turin, Turin, Italy*

²*Lamsum, Turin, Italy*

Correspondence should be addressed to Luca Ardito; luca.ardito@polito.it

Received 19 March 2020; Accepted 17 July 2020; Published 4 August 2020

Academic Editor: Daniela Breda

Copyright © 2020 Luca Ardito et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Software maintainability is a crucial property of software projects. It can be defined as the ease with which a software system or component can be modified to be corrected, improved, or adapted to its environment. The software engineering literature proposes many models and metrics to predict the maintainability of a software project statically. However, there is no common accordance with the most dependable metrics or metric suites to evaluate such institutional property. The goals of the present manuscript are as follows: (i) providing an overview of the most popular maintainability metrics according to the related literature; (ii) finding what tools are available to evaluate software maintainability and (iii) linking the most popular metrics with the available tools and the most common programming languages. To this end, we performed a systematic literature review, following Kitchenham's SLR guidelines, on the most relevant scientific digital libraries. The SLR outcome provided us with 174 software metrics, among which we identified a set of 15 most commonly mentioned ones, and 19 metric computation tools available to practitioners. We found optimal sets of at most five tools to cover all the most commonly mentioned metrics. The results also highlight missing tool coverage for some metrics on commonly used programming languages and minimal coverage of metrics for newer or less popular programming languages. We consider these results valuable for researchers and practitioners who want to find the best selection of tools to evaluate the maintainability of their projects or to bridge the discussed coverage gaps for newer programming languages.

1. Introduction

Nowadays, software security and resilience have become increasingly important, given how pervasive the software is. Effective tools and programming languages can

- (i) discover mistakes earlier
- (ii) reduce the odds of their occurrence
- (iii) make a large class of common errors impossible by restricting at compile time what the programmer can do

Several best practices are consolidated in software engineering, e.g., continuous integration, testing with code coverage measurement, and language sanitization. All these techniques allow the application of code analysis tools automatically, which can provide a significant enhancement of

the source code quality and allow software developers to efficiently detect vulnerabilities and faults [1]. However, the lack of comprehensive tooling may render it challenging to apply the same code analysis strategies to software projects developed with different languages or for different domains.

The literature defines software maintainability as the ease with which a software system or component can be modified to correct faults, improve performance or other attributes, or adapt to a changing environment [2]. Thus, maintainability is a highly significant factor in the economic success of software products. Several studies have described models and frameworks, based on software metrics, to predict or infer the maintainability of a software project [3–5]. However, although many different metrics have been proposed by the scientific literature over the course of the last 40 years, the available models are very language- and domain-

“The SLR outcome provided us with 174 software metrics, among which we identified a set of 15 most commonly mentioned ones, and 19 metric computation tools available to practitioners.”

Maintainability Index

@yegor256

References

Fred Brooks. *The Mythical Man-Month: Essays on Software Engineering*. Addison-Wesley Publishing Company, 1982. doi:[10.1002/spe.4380060417](https://doi.org/10.1002/spe.4380060417).

Don Coleman, Dan Ash, Bruce Lowther, and Paul Oman. Using Metrics to Evaluate Software System Maintainability. *Computer*, 27(8), 1994. doi:[10.1109/2.303623](https://doi.org/10.1109/2.303623).

Ward Cunningham. Experience Report — The WyCash Portfolio Management System. In *Proceedings of the Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)*, pages 29–30, 1992. doi:[10.1145/157710.157715](https://doi.org/10.1145/157710.157715).

Tim Gilboy. Maintainability Index — What Is It and Where Does It Fall Short? <https://sourcery.ai/blog/maintainability-index/>, 2022.

[Online; accessed 15-03-2024].

Tjaša Heričko. Exploring Maintainability Index Variants for Software Maintainability Measurement in Object-Oriented Systems. *Applied Sciences*, 2023. doi:[10.3390/app13052972](https://doi.org/10.3390/app13052972).

Rainer Niedermayr. Why We Don't Use the Software Maintainability Index. <https://teamscale.com/blog/en/news/blog/maintainability-index>, 2016. [Online; accessed 15-03-2024].

Paul Oman and Jack Hagemeister. Metrics for Assessing a Software System's Maintainability. In *Proceedings of the International Conference on Software Maintenance*, pages 337–338. IEEE, 1992. doi:[10.1109/icsm.1992.242525](https://doi.org/10.1109/icsm.1992.242525).

Arie van Deursen. Think Twice Before Using the “Maintainability Index”. <https://avandeursen.com/2014/08/29/think-twice-before-using-the-maintainability-index/>, 2014. [Online; accessed 15-03-2024].