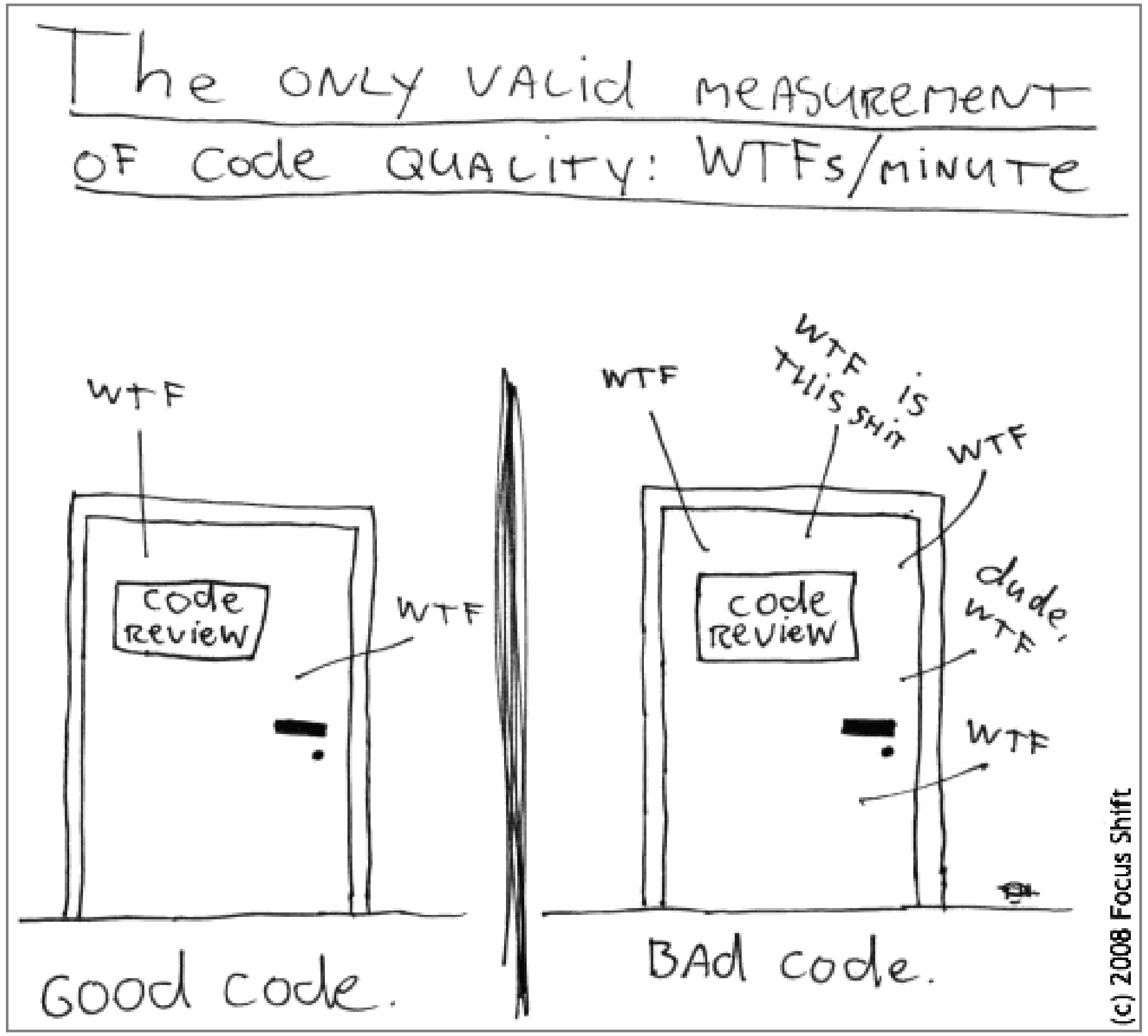
Maintainability Index

Yegor Bugayenko

Lecture #5 out of 24
80 minutes

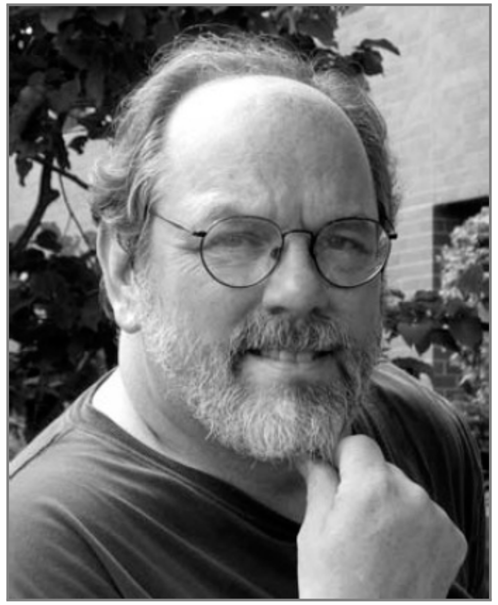
The slidedeck was presented by the author in this YouTube Video


The only valid measurement of code quality: WTFs/minute
WTF
code review
WTF
Good code.
WTF
WTF is this shit
WTF
code review
dude, WTF
WTF
BAd code.
(c) 2008 Focus Shift

Fred Brooks

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

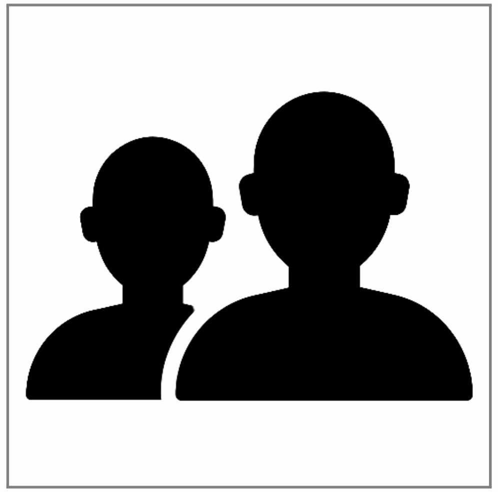
— Fred Brooks. *The Mythical Man-Month: Essays on Software Engineering.* Addison-Wesley Publishing Company, 1982. doi:10.1002/spe.4380060417



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."

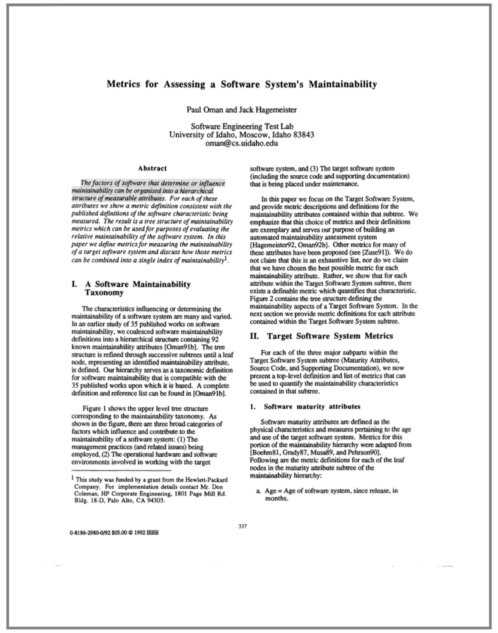
— Ward Cunningham. Experience Report — the WyCash Portfolio Management System. In *Proceedings of the Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)*, pages 29–30, 1992. doi:10.1145/157710.157715



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



Metrics for Assessing a Software System's Maintainability

Paul Oman and Jack Hagemeister

Software Engineering Test Lab
University of Idaho, Moscow, Idaho 83843
oman@cs.uidaho.edu

Abstract

The factors of software that determine or influence maintainability can be organized into a hierarchical structure of measurable attributes. For each of these attributes we show a metric definition consistent with the published definitions of the software characteristic being measured. The result is a tree structure of maintainability metrics which can be used for purposes of evaluating the relative maintainability of the software system. In this paper we define metrics for measuring the maintainability of a target software system and discuss how those metrics can be combined into a single index of maintainability[1].

I. A Software Maintainability Taxonomy

The characteristics influencing or determining the maintainability of a software system are many and varied. In an earlier study of 35 published works on software maintainability, we coalesced software maintainability definitions into a hierarchical structure containing 92 known maintainability attributes [Oman91b]. The tree structure is refined through successive subtrees until a leaf node, representing an identified maintainability attribute, is defined. Our hierarchy serves as a taxonomic definition for software maintainability that is compatible with the 35 published works upon which it is based. A complete definition and reference list can be found in [Oman91b].

Figure 1 shows the upper level tree structure corresponding to the maintainability taxonomy. As shown in the figure, there are three broad categories of factors which influence and contribute to the maintainability of a software system: (1) The management practices (and related issues) being employed, (2) The operational hardware and software environments involved in working with the target software system, and (3) The target software system (including the source code and supporting documentation) that is being placed under maintenance.

In this paper we focus on the Target Software System, and provide metric descriptions and definitions for the maintainability attributes contained within that subtree. We emphasize that this choice of metrics and their definitions are exemplary and serves our purpose of building an automated maintainability assessment system [Hagemeister92, Oman92b]. Other metrics for many of these attributes have been proposed (see [Zuse91]). We do not claim that this is an exhaustive list, nor do we claim that we have chosen the best possible metric for each maintainability attribute. Rather, we show that for each attribute within the Target Software System subtree, there exists a definable metric which quantifies that characteristic. Figure 2 contains the tree structure defining the maintainability aspects of a Target Software System. In the next section we provide metric definitions for each attribute contained within the Target Software System subtree.

II. Target Software System Metrics

For each of the three major subparts within the Target Software System subtree (Maturity Attributes, Source Code, and Supporting Documentation), we now present a top-level definition and list of metrics that can be used to quantify the maintainability characteristics contained in that subtree.

1. Software maturity attributes

Software maturity attributes are defined as the physical characteristics and measures pertaining to the age and use of the target software system. Metrics for this portion of the maintainability hierarchy were adapted from [Boehm81, Grady87, Musa89, and Pehrson90]. Following are the metric definitions for each of the leaf nodes in the maturity attribute subtree of the maintainability hierarchy:

a. Age = Age of software system, since release, in months.

[1] This study was funded by a grant from the Hewlett-Packard Company. For implementation details contact Mr. Don Coleman, HP Corporate Engineering, 1801 Page Mill Rd. Bldg. 18-D, Palo Alto, CA 94303.

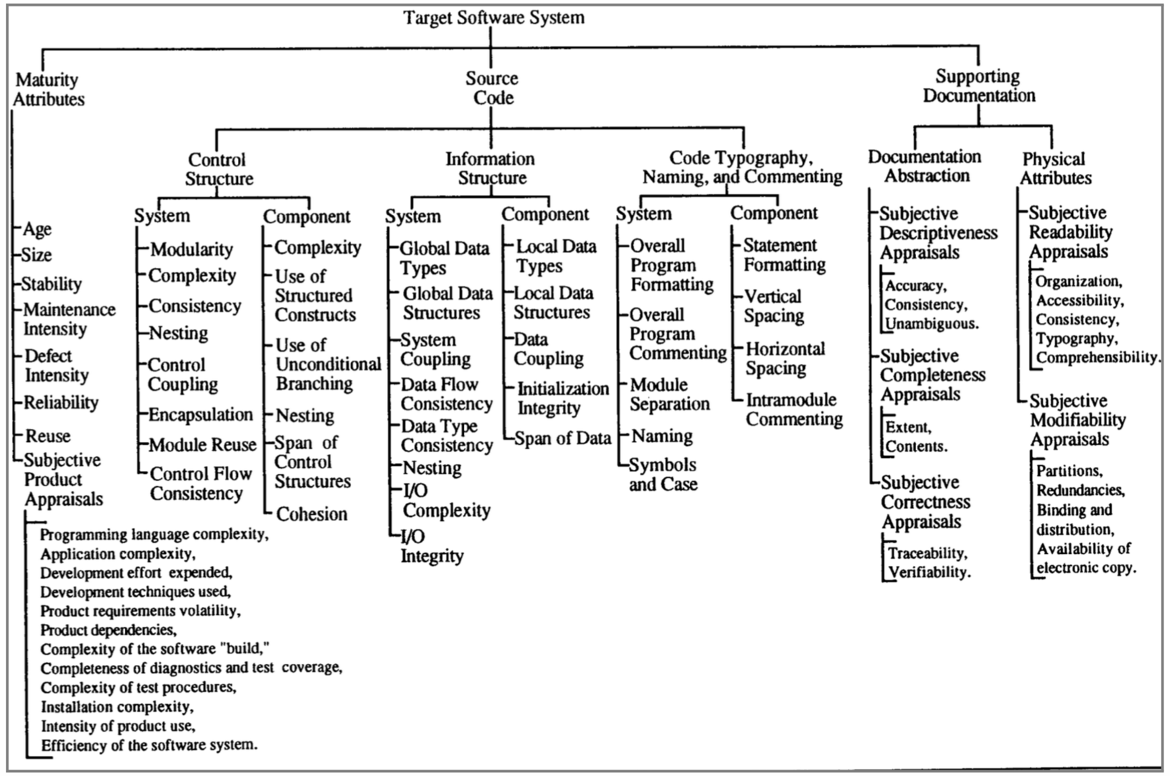
337

0-8186-2980-0/92 $03.00 © 1992 IEEE

"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

Software Maintainability Taxonomy



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

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

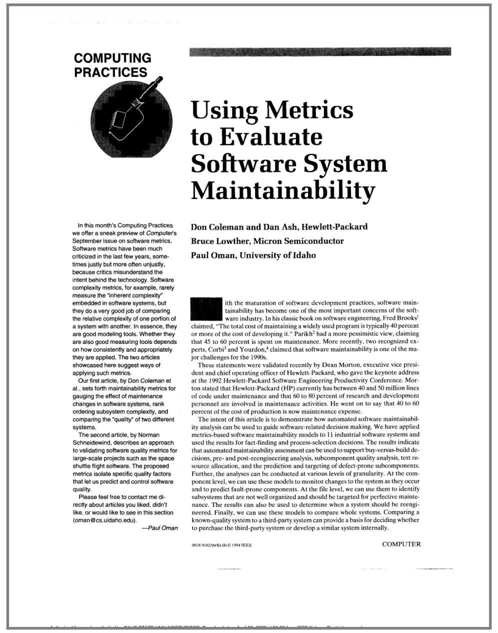
W_{D_i} = Weight of influence of maintainability Dimension D_i

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

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

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

"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."



COMPUTING PRACTICES

Using Metrics to Evaluate Software System Maintainability

Don Coleman and Dan Ash, Hewlett-Packard
Bruce Lowther, Micron Semiconductor
Paul Oman, University of Idaho

In this month's Computing Practices we offer a sneak preview of *Computer*'s September issue on software metrics. Software metrics have been much criticized in the last few years, sometimes justly but more often unjustly, because critics misunderstand the intent behind the technology. Software complexity metrics, for example, rarely *measure the "inherent complexity"* embedded in software systems, but they do a very good job of comparing the relative complexity of one portion of a system with another. In essence, they are good modeling tools. Whether they are also good measuring tools depends on how consistently and appropriately they are applied. The two articles showcased here suggest ways of applying such metrics.

Our first article, by Don Coleman et al., sets forth maintainability metrics for gauging the effect of maintenance changes in software systems, rank ordering subsystem complexity, and comparing the "quality" of two different systems.

The second article, by Norman Schneidewind, describes an approach to validating software quality metrics for large-scale projects such as the space shuttle flight software. The proposed metrics isolate specific quality factors that let us predict and control software quality.

Please feel free to contact me directly about articles you liked, didn't like, or would like to see in this section (oman@cs.uidaho.edu).

—*Paul Oman*

[illegible]ith the maturation of software development practices, software maintainability has become one of the most important concerns of the software industry. In his classic book on software engineering, Fred Brooks[1] claimed, "The total cost of maintaining a widely used program is typically 40 percent or more of the cost of developing it." Parikh[2] had a more pessimistic view, claiming that 45 to 60 percent is spent on maintenance. More recently, two recognized experts, Corbi[3] and Yourdon,[4] claimed that software maintainability is one of the major challenges for the 1990s.

These statements were validated recently by Dean Morton, executive vice president and chief operating officer of Hewlett-Packard, who gave the keynote address at the 1992 Hewlett-Packard Software Engineering Productivity Conference. Morton stated that Hewlett-Packard (HP) currently has between 40 and 50 million lines of code under maintenance and that 60 to 80 percent of research and development personnel are involved in maintenance activities. He went on to say that 40 to 60 percent of the cost of production is now maintenance expense.

The intent of this article is to demonstrate how automated software maintainability analysis can be used to guide software-related decision making. We have applied metrics-based software maintainability models to 11 industrial software systems and used the results for fact-finding and process-selection decisions. The results indicate that automated maintainability assessment can be used to support buy-versus-build decisions, pre- and post-reengineering analysis, subcomponent quality analysis, test resource allocation, and the prediction and targeting of defect-prone subcomponents. Further, the analyses can be conducted at various levels of granularity. At the component level, we can use these models to monitor changes to the system as they occur and to predict fault-prone components. At the file level, we can use them to identify subsystems that are not well organized and should be targeted for perfective maintenance. The results can also be used to determine when a system should be reengineered. Finally, we can use these models to compare whole systems. Comparing a known-quality system to a third-party system can provide a basis for deciding whether to purchase the third-party system or develop a similar system internally.

0018-9162/94/$4.00 © 1994 IEEE

"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

$$\begin{aligned}\text{Maintainability} &= 171 \\ &- 3.42 \times \ln(aveE) \\ &- 0.23 \times aveV(g') \\ &- 16.2 \times \ln(aveLOC) + aveCM\end{aligned}$$

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

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

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 Maintainabillity
△	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]

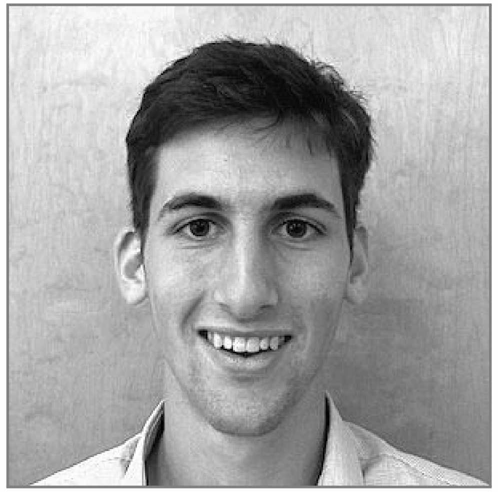
> “The Maintainability Index does not provide information about the impact on development activities. A value of 57 does not express which maintainability aspects are affected by a bad value.” — Rainer Niedermayr

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

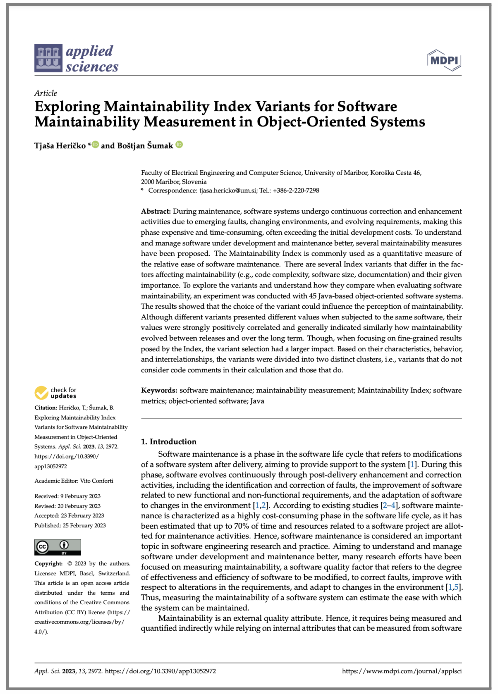


Tim Gilboy

"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



applied sciences

MDPI

Article

Exploring Maintainability Index Variants for Software Maintainability Measurement in Object-Oriented Systems

Tjaša Heričko * and Boštjan Šumak

Faculty of Electrical Engineering and Computer Science, University of Maribor, Koroška Cesta 46, 2000 Maribor, Slovenia
* Correspondence: tjasa.hericko@um.si; Tel.: +386-2-220-7298

Abstract: During maintenance, software systems undergo continuous correction and enhancement activities due to emerging faults, changing environments, and evolving requirements, making this phase expensive and time-consuming, often exceeding the initial development costs. To understand and manage software under development and maintenance better, several maintainability measures have been proposed. The Maintainability Index is commonly used as a quantitative measure of the relative ease of software maintenance. There are several Index variants that differ in the factors affecting maintainability (e.g., code complexity, software size, documentation) and their given importance. To explore the variants and understand how they compare when evaluating software maintainability, an experiment was conducted with 45 Java-based object-oriented software systems. The results showed that the choice of the variant could influence the perception of maintainability. Although different variants presented different values when subjected to the same software, their values were strongly positively correlated and generally indicated similarly how maintainability evolved between releases and over the long term. Though, when focusing on fine-grained results posed by the Index, the variant selection had a larger impact. Based on their characteristics, behavior, and interrelationships, the variants were divided into two distinct clusters, i.e., variants that do not consider code comments in their calculation and those that do.

Keywords: software maintenance; maintainability measurement; Maintainability Index; software metrics; object-oriented software; Java

check for updates

Citation: Heričko, T.; Šumak, B. Exploring Maintainability Index Variants for Software Maintainability Measurement in Object-Oriented Systems. *Appl. Sci.* **2023**, *13*, 2972. https://doi.org/10.3390/app13052972

Academic Editor: Vito Conforti

Received: 9 February 2023
Revised: 20 February 2023
Accepted: 23 February 2023
Published: 25 February 2023

1. Introduction

Software maintenance is a phase in the software life cycle that refers to modifications of a software system after delivery, aiming to provide support to the system [1]. During this phase, software evolves continuously through post-delivery enhancement and correction activities, including the identification and correction of faults, the improvement of software related to new functional and non-functional requirements, and the adaptation of software to changes in the environment [1,2]. According to existing studies [2–4], software maintenance is characterized as a highly cost-consuming phase in the software life cycle, as it has been estimated that up to 70% of time and resources related to a software project are allotted for maintenance activities. Hence, software maintenance is considered an important topic in software engineering research and practice. Aiming to understand and manage software under development and maintenance better, many research efforts have been focused on measuring maintainability, a software quality factor that refers to the degree of effectiveness and efficiency of software to be modified, to correct faults, improve with respect to alterations in the requirements, and adapt to changes in the environment [1,5]. Thus, measuring the maintainability of a software system can estimate the ease with which the system can be maintained.

Maintainability is an external quality attribute. Hence, it requires being measured and quantified indirectly while relying on internal attributes that can be measured from software

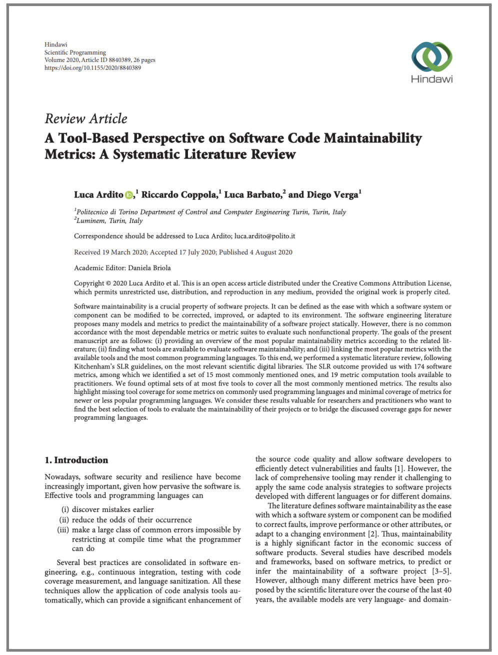
Tjaša Heričko

"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

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 2020, Article ID 8840389, 26 pages
https://doi.org/10.1155/2020/8840389

Hindawi

Review Article

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

Luca Ardito,[1] Riccardo Coppola,[1] Luca Barbato,[2] and Diego Verga[1]

[1] *Politecnico di Torino Department of Control and Computer Engineering Turin, Turin, Italy*
[2] *Luminem, 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 Briola

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 nonfunctional 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."

—

References

Fred Brooks. *The Mythical Man-Month: Essays on Software Engineering*. Addison-Wesley Publishing Company, 1982. doi: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.

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.

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.

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.

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].