Modified Delta Maintainability Model of Object-Oriented Software

Pavlo Skladannyi[1], Olena Nehodenko[2], Svitlana Shevchenko[1], Oksana Zolotukhina[2], and Vitalii Nehodenko[2]

[2] *Borys Grinchenko Kyiv University, 18/2 Bulvarno-Kudryavska str., Kyiv, 04053, Ukraine*
[1] *State University of Telecommunications, 7 Solomenska str., Kyiv, 03110, Ukraine*

Abstract

Modern software systems are increasingly integrated into vital society areas, from managing critical infrastructure to piloting vehicles. That is why one of the most important priorities is the reduction of possible software defects. The speed of development of social processes and technologies determines the need for adaptation, which in turn requires software adjustments. Analysis of various definitions and aspects of maintainability, as well as established models and approaches to measuring object-oriented software, remains a relevant issue. This analysis makes it possible to determine the possibilities of improving the efficiency of the assessment by methods of statistical analysis. Predictive assumptions about object development include maintainability of object-oriented software. At the same time, the method of modifying Delta Maintainability Model (DMM) by expanding the measurable properties of the source code is used. It is important to demonstrate the stability and effectiveness of object-oriented software change measurement by conducting comparative analysis for source code changes, which makes it possible to measure maintainability in processes with continuous delivery and uninterrupted integration methodological approaches. At the same time, the interpretation of the assessment results makes it possible to establish a causal relationship and eliminate shortcomings.

Keywords

Quality of software, delta maintainability model, DMM, SIG maintainability model, SIG-MM, object-oriented software, methods of statistical analysis, Mozilla Rhino.

1. Introduction

Research by C. Jones shows a steady increase in the involvement of engineers in software support jobs, from 9.09% percent in 1950 to 72.73% in 2000 and a projected involvement rate of 77.27% in 2025 [1]. The requirements to increase the level of quality and maintainability are the reason for many studies and the constant search for software measurement and evaluation methodologies. Software maintainability is a well-researched topic [2].

Most of the research is based on the importance of software maintainability, emphasizing the relationship with adaptability, and reducing the number of defects [3]. The quality of the software, specifically, maintainability differs from other properties by the complexity of the factors that determine them, so their adjustment is almost always difficult and long-term [4].

Scientific studies devoted to software quality problems include publications of authors: C. Jones [1], R. Plösch, H. Gruber, C. Korner, M. Saft [5], A. Madi, O.K. Zein [6].

Improving the quality and maintainability of object-oriented software by reducing the number of defects is important for any development.

CPITS-2022: Cybersecurity Providing in Information and Telecommunication Systems, October 13, 2022, Kyiv, Ukraine
EMAIL: p.skladannyi@kubg.edu.ua (P. Skladannyi); negodenkoav@i.ua (O. Nehodenko); s.shevchenko@kubg.edu.ua (S. Shevchenko); zolotukhina.oks.a@gmail.com (O.Zolotukhina) negodenkovp@gmail.com (V. Nehodenko);
ORCID: 0000-0002-7775-6039 (P. Skladannyi); 0000-0001-6645-1566 (O. Nehodenko); 0000-0002-9736-8623 (S. Shevchenko); 0000-0002-3314-417 (O. Zolotukhina); 0000-0002-7678-9138 (V. Nehodenko)

2. Maintainability in Modern Software Quality Models

Software quality models define maintainability as one of the main attributes. Most of the methods of measuring maintainability, consider not only maintainability itself, but are also part of quality models. Maintainability is a complex concept that has been repeatedly examined by studies offering different formulations and measurement approaches. However, most of the studied approaches, to some extents, have the following aspects:

- Ambiguity of terminology and definitions of quality criteria.
- Abstractness, absence of definitions of formulations and methods of measurement.
- Complexity or impossibility of interpretation.
- Conducting a causal analysis of measurement results.

McCall's model [7] consists of a 3-level hierarchy and determines the relationship between software quality attributes. Maintainability, as an internal quality factor, is related to product modification and is defined as the amount of effort required to identify and correct a program defect in the operating environment. Maintainability, like all internal quality factors, is measured indirectly through related software properties: simplicity, brevity, informativeness, and modularity.

It is offered to measure the properties by ranking from 1 (goal not achieved) to 10 (excellent implementation), without indicating specific metrics or methods of measurement. An example of the Fig. 1 according to the model, maintainability is determined by the levels of the hierarchy.

Quality Model for Object-oriented Design (QMOOD), offered by Bansia and Davis in 2002 [8]. This model consists of a 4-level hierarchical structure. The model also includes a set of metrics designed to measure quality attributes. The definition of quality attributes, with some modifications, is based on the software quality model ISO 9126 [9]. The model also defines design attributes and their corresponding metrics. Design attributes, in turn, are related to quality attributes. Maintainability, which is defined by ISO 9126 as a quality attribute, implies a certain stage of software completion, therefore the model is focused only on its sub-characteristic—understandability, which should allow using the model at earlier stages of development.

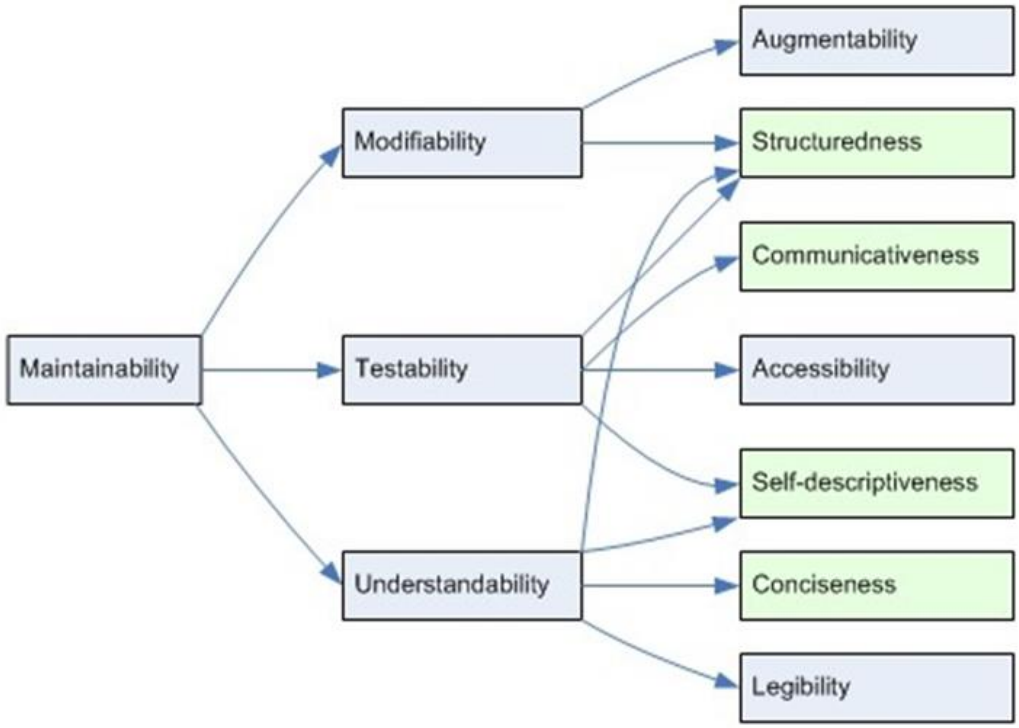

Figure 1: Maintainability model hierarchy levels

The connections between design properties and quality attributes [8] in the model (QMOOD) were established. Basic metrics for measuring design properties can be defined.

Design Size in Classes (DSC) is the number of all classes provided by the program design.

Number of Hierarchies (NOH) is the amount of class hierarchies of the program.

Average Number of Attributes (ANA) is the average of the number of classes inherited by a class and is calculated by counting the number of classes along all inheritance paths from the root class(es) to all classes of the inheritance structure.

Data Access Metric (DAM) is a value between 0 and 1 and is the ratio of the number of private (protected) attributes of a class to the total number of attributes defined by the class.

Direct Class Coupling (DCC) is the number of classes on which the class depends directly, either by defining an attribute or passing message (parameters) in methods.

Class Association Method (CAM) is a value in the range from 0 to 1, and is a calculation of the interdependence of class methods based on the list of method parameters [10]. The metric is calculated by summing the intersection of the method parameters with the maximally independent set of parameters of all types in the class.

Measure of Aggregation (MOA) this metric measures the degree of part-to-whole connection implemented by attributes. The value is the sum of the number of declared data whose types are user-defined classes.

Measure of Functional Abstraction (MFA) ranges from 0 to 1 and is the ratio of the number of methods inherited by a class to the total number of methods accessed from other methods of the class.

Number of Polymorphic Methods (NOP) is the sum of the number of methods that can exhibit polymorphic behavior.

Class Interface Size (CIS) is a quantitative measure of the count of public methods of a class.

Number of Methods (NOM) is a quantifier of the count of all methods defined by a class.

The quality model ISO/IEC 25010:2016 "Systems and software engineering" defines the Security Quality Requirements Engineering (SQuaRE). System and software quality models [11, 12] is a replacement for the ISO 9126 standard [9] (Fig. 2). The standard extends the quality model with two new high-level characteristics: compatibility (is a new characteristic) and security (in the previous standard is a sub-characteristic of functional suitability).

Table 1

Matrix of relationships of design properties and quality attributes

Attributes of design	**Attribute of quality**					
	Reusability	Flexibility	Understan-dability	Functionality	Expandability	Efficiency
Size of design	+	–	–	+	–	–
Hierarchy	–	–	–	+	–	–
Abstractness	–	–	–	–	+	+
Encapsulation	–	+	+	–	–	+
Connectivity	–	–	–	–	–	–
Connectedness	+	–	+	+	–	–
Composition	–	+	–	–	–	+
Inheritance	–	–	–	–	+	+
Polymorphism	–	+	–	+	+	+
Message exchange	+	–	–	+	–	–

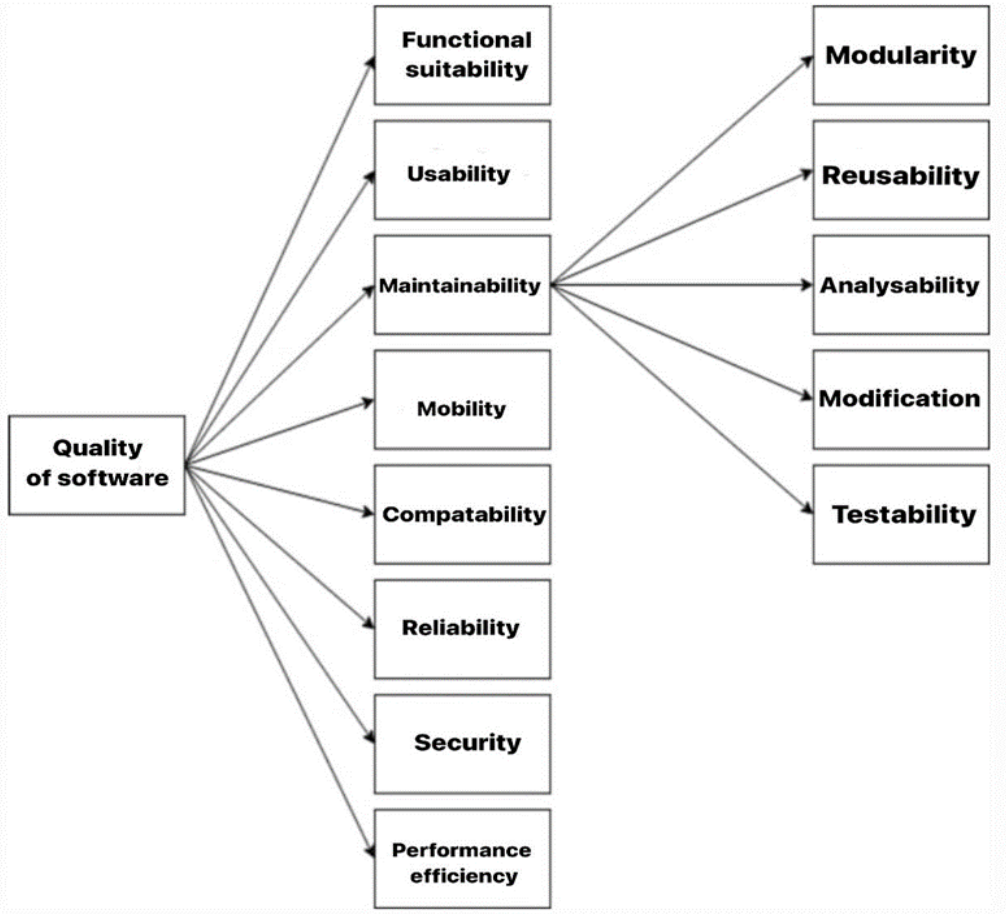

Figure 2: High-level characteristics of product quality according to ISO/IEC 25010

The standard also introduced changes to the maintainability structure. New sub-characteristics have been introduced: "modularity" and "reusability." The subcharacteristics "variability" and "stability" were replaced by the new characteristic "modification." The standard provides the following definitions of maintainability:

- Maintainability is defined as the degree of effectiveness and efficiency in which a product or system can be modified by designated service personnel. At the same time, maintainability can be construed as the inherent ability of a product or system to facilitate maintenance work or as quality during use;
- Modularity is the degree to which a system or computer program is composed of discrete components such that a change in one component has minimal impact on other components.
- Reusability is the degree to which an asset can be used in more than one system or in the construction of other assets.
- Analysability is defined as the degree of effectiveness and efficiency in which it is possible to assess the effect on the product or system of the intended change in one or more parts, or to diagnose non-completion, the causes of failures, or to identify the parts to be modified.
- Modification is the degree of effectiveness and efficiency in which a product or system can be modified without introducing defects or degrading the existing quality of the product. Modification is a combination of variability and stability.
- Testability is the degree of effectiveness and efficiency to which test criteria can be established for a system, product, or component, and tests must be performed to determine whether those criteria have been met.

3. Maintainability Model (SIG-MM)

The model is independent of the programming language and software

architecture, has indicators that are easy to understand and explain, and is based on defined relationships of system-level quality characteristics defined by the ISO 9126-1 standard [9] with the characteristics of the source code properties and their metric indicators. To a large extent, this approach is based on the need to identify causal relationships between source code properties and maintainability, because the latest is a complex and multi-component quality attribute.

The simple symbolic scale ++ / + / o / – / – – is used to rank the results of evaluating a particular property of the code.

The size of the program is one of the simple and direct indicators of maintainability, because the larger the size, the more effort is required for cognitive perception, making changes, testing.

The indicator is set for each language separately on the basis of research [13] that determine the relationship between the average number of lines of code (LOC) in a separate programming language per one functional point and the number of functional points that can be produced by one person in one month.

For the purposes of the model under consideration, the size of the program is determined by the person-years required to create the program (Table 2).

Table 2

Ranking of the evaluation results of a separate property of the code

Rank	The number of person-years
++	0–8
+	8–30
o	30–80
–	80–160
—	> 160

Thus, a software product requiring 160 person-years is considered too large, and for systems implemented in Java is equal to 1.3 million lines of code or 2.6 for the COBOL language.

The complexity per unit of the program is a property of the source code and is defined as the degree of internal complexity of the units of the source code that it consists of (Table 3).

Table 3

Cyclomatic complexity of the program

Cyclomatic complexity	Definition of risk
1–10	Simple
11–20	Complicated, moderate risk
21–50	Complex, high risk
> 50	Absence of testing, very high risk

With further aggregation of complexity per program unit to determine the ratio of lines of code of each risk level as a percentage. That is, if the program consists of 20,000 lines of code, and at the same time the sum of the lines of code of the program units with a high risk of complexity is 1,000 lines of code, the aggregate value for the high risk category will be 5% (Table 4).

Table 4

Volume with the final distribution relative to different risk levels for system ranking

Rank	The maximum relative amount of lines of code (LOC), %		
	Moderate risk	High risk	Very high risk
++	25	0	0
+	30	5	0
o	40	10	0
–	50	15	5
—	—	—	—

Thus, for example, if the rank of the program is defined as "+", the amount of lines of code with high risk does not exceed 5%, with high risk 15% and 50% of lines of code are within the limits of moderate risk.

Duplication (or cloning of code) reduces the cognitive perception of the program, the possibility of making changes, and unmotivatedly increases the size of the program. And it is defined as the repetition of a block of code for more than 6 lines, while spaces at the beginning of the lines are not taken into account to determine the repetition (Table 5).

Table 5
Programs duplication parameters

Rank	Duplication, %
++	0–3
+	3–5
o	5–10
–	10–20
—	20–100

The size of the program unit is an important indicator, because large-sized program units require more costs to support, also, this indicator additionally indirectly displays the possible complexity. The indicator is defined as the number of lines of code (LOC) followed by size categorization and ranking similar to the definition of complexity per program unit.

Module testing is calculated as a relative indicator of program coverage by program unit tests. This practice is not static analysis and refers to dynamic code analysis.

The calculation of the overall assessment of the system is carried out by the average weighting of the indicators of each property of the source code.

For example: program size is rated as small "++", with very high complexity per program unit "—", high duplication and program unit size "–", and moderate testing. Accordingly, the analyzability of such software as well as the stability are average, while variability and testability are low, which is averaged by a low "–" maintainability assessment. However, these results are more effective in determination of causal relationship. Thus, in order to increase maintainability, it is necessary to carry out refactoring aimed at program units of very high complexity with the aim of reducing it, and reducing the size of the program units, as well as removing duplications.

4. A modified Model of Delta Maintainability of Object-Oriented Software

The Maintainability Model (SIG-MM) does not have certain disadvantages, but it is based on the measurement of the entire source code of the software product, which causes the weak representativeness of the measurement results with minor changes in the source code.

An example of this is entry #402331 in the bug registration system of the Mozilla Rhino software product [14]. The specified defect was fixed by commit #262602 [15], which is in the measurements range from –5 to 5. The maintainability model (SIG-MM) has a rating of –0.007. The specified result does not show any significant changes in maintainability, which is not true, because the 200 lines of code introduced by the changes have a significant negative impact on maintainability. However, in relation to the size of all Mozilla Rhino code terms, compared to which the changes were validated, the indicator received an unrepresentative result [16].

There is a Delta Maintainability Model (DMM), which does not contain the above shortcomings, but does not take into account the specifics of the object-oriented programming paradigm. It is intended to compare and analyze partial changes to the source code, not the program as a whole. The model integrates with version control systems, allowing integration with DevOps tools for use in analyzing ongoing source code assessments. Risk ranking is based on the threshold values of the Maintainability Model (SIG-MM) [17].

This article proposes a modification of the delta Maintainability Model (DMM) by expanding the measurable properties of the source code, determining their relationship with the sub-characteristics of maintainability defined by ISO/IEC 25010:2016, and determining measurement methods and threshold values [11, 12].

Further the basic model is denoted by DMM, the resulting indicator is DMMS. References to the proposed model are denoted as DMM+, the indicator, respectively, DMMS+

The calculation takes place taking into account the order established for the Delta Maintainability Model (DMM) [18] with the following changes in definitions:

RC = {low, high};

CP = {class Connectivity, class Difficulty, method Difficulty, the Number of methods, method Size, the Number of parameters, Dublication, module Dependency}.

Philo, Tarsio G.S., and M. Bigonya [19] conducted a research of 111 software systems and proposed the determination of threshold values of object-oriented software metrics (Table 6).

The practical validation of the proposed model was carried out by analyzing software

products with open source code, implemented using an object-oriented programming paradigm, a development process using version control systems and a significant number of participants in the development process and a long history of code changes.

Table 6
Software metrics Philo, Tarsio G.S., and M. Bigonya

Metrics	Level		
	Better	Medium	Bad
WMC	m ≤ 11	11 < m ≤ 34	m > 34
NOC	m ≤ 11	11 < m ≤ 28	m > 28
NOM	m ≤ 6	6 < m ≤ 14	m > 14
MLOC	m ≤ 10	10 < m ≤ 30	m > 30
PAR	m ≤ 2	2 <m ≤ 4	m> 4
VG	m ≤ 2	2 <m ≤ 4	m> 4

Because the analysis of the source code of software products requires the study of the abstract syntax tree (AST), therefore, in order to simplify the implementation of the application intended for analysis, all software products are selected with the requirement of implementing the object-oriented part with a common programming language.

For the comparative analysis, six software products of different functional purposes, with open source code, with the implementation of the object-oriented part in the Python programming language, were chosen.

The scope of analysis covers only changes in files (modules) that contain instructions in the Python programming language and have the extension "*.py", while changes to files with instructions for unit testing are not taken into account.

In order to carry out research and measurements, a program was implemented with command line interface support and simultaneous measurement of DMM and DMM+ model indicators.

According to the results of the analysis, a significant indicator of the Pearson correlation coefficient ranging from 0.77 to 0.86 demonstrates a strong positive correlation between DMMS and DMMS+ values. Thus, DMMS+ metrics along with DMMS reflect the relationship to source code changes that affect maintainability. Values of DMMS indicators and validation were confirmed [20] in the course of empirical research.

Correlation of DMMS and DMMS+ indicators according to the analysis of 1000 changes made in the repository:

1. Tensorflow $r = 0.86$.
2. Sentry $r = 0.8$.
3. Django $r = 0.82$.
4. Odoo $r = 0.84$.
5. Saleor $r = 0.77$.
6. Zulip $r = 0.77$.

Despite the positive correlation, the indicators show fluctuation according to the indicator of the absolute average difference. It is important to show the changes in the indicator of the absolute average difference between the indicators of DMMS and DMMS+ based on the results of the analysis of 1000 changes made to each of the repositories (Fig. 3).

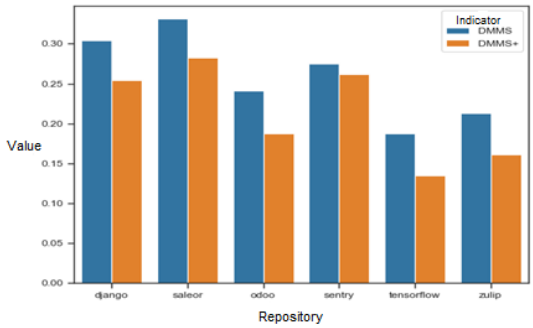

Figure 3: Illustration of changes in absolute mean difference between DMMS and DMMS+

5. Conclusions

Summarizing the above considerations, it should be noted that in this work, the analysis of existing models and measurement metrics of object-oriented software was carried out, their advantages and disadvantages were determined.

A modified delta model of object-oriented software maintainability, by expanding the measurable properties of the source code, determining their relationship with the maintainability sub-characteristics defined by ISO/IEC 25010:2016. Measurement methods and threshold values are defined. Practical validation of the offered model was carried out by analyzing open source software products implemented using an object-oriented programming paradigm, a development process using version control systems and a significant

number of participants in the development process and a long history of code changes.

The stability and effectiveness of measuring changes in object-oriented software has been demonstrated by means of a comparative analysis of changes made to the source code, which makes it possible to measure compliance in processes with methodological approaches of continuous delivery and uninterrupted integration. At the same time, the interpretation of the evaluation results makes it possible to establish a causal relationship and eliminate shortcomings.

6. References

[1] C. Jones, The Economics of Software Maintenance in the Twenty First Century, 2006.

[2] V. Grechaninov, et al., Decentralized Access Demarcation System Construction in Situational Center Network, in Workshop on Cybersecurity Providing in Information and Telecommunication Systems II, vol. 3188, no. 2, 2022, pp. 197–206.

[3] V. Buriachok, V. Sokolov, P. Skladannyi Security Rating Metrics for Distributed Wireless Systems, in 8th International Conference on "Mathematics. Information Tech-nologies. Education," vol. 2386, 2019, pp. 222–233.

[4] Kipchuk, F., et al., Investigation of Availability of Wireless Access Points based on Embedded Systems, in IEEE International Scientific-Practical Conference Problems of Infocommunications, Science and Technology, 2019, pp. 246–250. doi: 10.1109/picst47496.2019.9061551.

[5] The Cost of Poor Software Quality in the US: A 2020 Report: The Consortium for Information & Software Quality (CISQ) 4.

[6] A. J. Albrecht, Measuring Application Development Productivity, in Proceedings of the Joint SHARE, GUIDE, and IBM Application Development Symposium, 1979, pp. 83–92.

[7] J. McCall, P. Richards, G. Walters, Factors in Software Quality, vol. III, Preliminary Handbook on Software Quality for an Acquisiton Manager.

[8] V. R. Basili, L. C. Briand, W. L. Melo, A validation of Object-Oriented Design Metrics as Quality Indicators, IEEE Transactions on Software Engineering, vol. 22, no. 10, 1996, pp. 751–761.

[9] ISO/IEC 9126-2001. Software Engineering. Product Quality 1, Quality Model.

[10] J. Bansiya, C. Davis, Class Cohesion Metric For Object-Oriented Designs, J. Object-Oriented Programming, vol. 11, no. 8, 1999, pp. 47–52.

[11] S. Cohen, W. Nutt, Y. Sagic, Deciding Equivalances Among Conjunctive Aggregate Queries, J. ACM 54, 2007. doi: 10.1145/1219092.1219093.

[12] ISO/IEC 25010:2011 Systems and Software Engineering. Systems and Software Quality Requirements and Evaluation. System and Software Quality Models.

[13] ISO/IEC 25010:2016 Systems and Software Engineering. Requirements for the Quality of Systems and Software Tools and its Evaluation (SQuaRE). Models of system and software quality` (in Ukrainian).

[14] Rhino Graveyard. Bug 402331, https://bugzilla.mozilla.org/show_bug.cgi?id=402331.

[15] Mozilla / Rhino. Fix bug 402331, https://github.com/mozilla/rhino/commit/262602.

[16] Software Productivity Research LCC, Programming Languages Table, ver. 2006b, 2006.

[17] M. di Biase, et al., The Delta Maintainability Model: Measuring Maintainability of Fine-Grained Code Changes, in 2019 IEEE/ACM International Conference on Technical Debt (TechDebt), 2019, pp. 113–122. doi: 10.1109/TechDebt.2019.00030.

[18] I. Heitlager, T. Kuipers, J. Visser, A Practical Model for Measuring Maintainability, in 6th International Conference on the Quality of Information and Communications Technology, 2007, pp. 30–39. doi: 10.1109/QUATIC.2007.8.

[19] M. di Biase, et al., The Delta Maintainability Model: Measuring Maintainability of Fine-Grained Code Changes Technical Report, J. Cohen (Ed.), Special issue: Digital Libraries, vol. 39, 1996.

[20] T. G. S. Filó, M. Bigonha. A Catalogue of Thresholds for Object-Oriented Software Metrics, 2015.