UDC 004.05
doi:10.31799/1684-8853-2023-1-17-28
EDN: ORVZMP

Articles

Advanced metric analysis tool for Java source code

V. V. Burakov[a], *Dr. Sc., Tech., Associate Professor, orcid.org/0000-0002-0158-8681, burakov@compmechlab.com*
A. I. Borovkov[b], *PhD, Tech., Professor, orcid.org/0000-0003-3177-0959*
[a]*CompMechLab LLC, bld. 2a, 21, Gzhatskaya St., 195220, Saint-Petersburg, Russian Federation*
[b]*Peter the Great St. Petersburg Polytechnic University, 29, Politekhnicheskaia St., 195251, Saint-Petersburg, Russian Federation*

Introduction: *Despite considerable efforts made by numerous researchers and developers, procedures for software quality assessment are still not sufficiently formalized and automated.* ***Purpose:*** *To develop a specialized software tool to quantify the structural properties of Java code.* ***Results:*** *We have developed MetricsTree, a software tool that calculates 61 established object-oriented metrics and is one of the largest sets among similar tools. MetricsTree is integrated into the IDE to ensure the fastest possible information delivery, contains unique visualization tools to increase the efficiency of metrics analysis, and implements a metrics profile mechanism to select classes based on a set of metrics values.* ***Practical relevance:*** *As a result of applying MetricsTree to automate quality assurance processes in the development of Peter the Great St.Petersburg Polytechnic University's flagship CML-Bench system (a platform for developing digital twins), the average number of software defects detected by external means decreased by 34% during the year.*

Keywords *— software quality, software metrics, object-oriented metrics, Java metrics, software metrics tool, metrics analysis, software metrics visualization, code smell.*

For citation: Burakov V. V., Borovkov A. I. Advanced metric analysis tool for Java source code. *Informatsionno-upravliaiushchie sistemy* [Information and Control Systems], 2023, no. 1, pp. 17–28. doi:10.31799/1684-8853-2023-1-17-28, EDN: ORVZMP

Introduction

The digital platform for development and application of CML-Bench for digital twins has been created and continues to be maintained by specialists at the STI Centre "New Manufacturing Technologies" at Peter the Great St. Petersburg Polytechnic University (https://cml-bench.ru). Digital twins [1], which our system manages, are families of complex multidisciplinary mathematical and computer models with a high level of adequacy to real materials, objects, structures, machines, devices, technical systems, physico-mechanical, technological and production processes [2, 3]. CML-Bench is a system for numerical design, mathematical modelling and computer-aided engineering control that belongs to the SPDRM (Simulation, Process, Data & Recourses Management) class of systems, providing standardized computations and processing of results, coordination of distributed engineering and design teams, greater transparency of design and engineering processes, optimized development of digital twins through an accumulated knowledge base and intelligent supercomputer management.

CML-Bench has been developed since 2014, including more than 600,000 lines of code; it comprises a set of services whose server side is implemented in Java. CML-Bench is implemented at enterprises from the high-tech industries with stringent requirements to code quality.

As initial steps, we adopted a regulation that we built for quality assurance, describing the relevant procedures and tools. Initially, we used SonarQube (https://www.sonarqube.org) and PMD (https://pmd.github.io) as basic tools.

Problem statement

The importance of the tasks solved by software implemented in all areas of human activity, the possible damage from the exploitation of low-quality software is very high, so it is very important to formalize all components of software quality assessment. A set of specialized mathematical models has been developed to solve this problem [4–6]. One way to formalize the quality assessment process is based on the calculation and interpretation of object-oriented values of source code metrics. Researches on source code metric analysis started in the 80-s, at present several tens of metrics types are used, the correlation of their values with the qualitative state of software is scientifically proved, and there are several tens of software applications, designed for calculation of Java code metrics. We investigated this software as part of selecting a calculation tool to implement in our quality assurance processes. After determining the functionality requirements needed for our processes, we identified weaknesses in the existing metrics calculation software that made implementation difficult. We concluded that the existing products primarily lacked the following important features:

— support for some types of metrics;

— possibility to calculate new metrics with minimum labor intensity;
— rapid data delivery;
— integration into IDE;
— possibility to investigate samples of classes corresponding to a given set of metrics values.

As a solution to these problems, we developed a plugin for the IntelliJ IDEA IDE, which we called MetricsTree.

Practical implementation

MetricsTree is a plugin for IntelliJ IDEA (https://plugins.jetbrains.com/plugin/13959-metricstree) that is compatible with the Ultimate, Community, Educational, and Android Studio editions. Plugin development for IDEA consists in correlating plugin classes with special extension points provided by this IDE to extend functionality. As a result, classes become available to the elements of the IDE from the methods of the plugin; the IDE then determines from the extension points where to delegate the corresponding calls in response to user actions, events of the plugin or the IDE itself. The Java code model generated by IDEA and used by MetricsTree is the PSI (Program Structure Interface) tree, which is similar to the AST (Abstract Syntax Tree) in this IDE. This makes it possible to feed the model of program elements of the source Java file to MetricsTree input as efficiently as it is implemented in IDEA itself.

MetricsTree accesses Java code elements in the form of a PSI-tree, performs their calculation and collects the value of a particular metric. Two hierarchies of classes, for metrics and for code elements, are used to model the subject domain. Hierarchies of classes have been developed that are responsible for sequential traversal of PSI-tree nodes and calculate them according to the rules defined by the metrics, which are traditionally represented as the Visitor pattern for this kind of problems.

The source code of MetricsTree (https://github.com/b333vv/metricstree) is freely distributed under the Apache 2.0 license and is structured, clean, simple and straightforward to the extent that there is no need for more detailed descriptions of its technical implementation in this article.

Supported metrics

We analyzed the most cited research on object-oriented metrics, some of them were related to coupling metric analysis [7–10], the other parts is with maintenance [11], reusability [12, 13] and complexity [14, 15] and chose a set of metrics to implement in our plugin. The set turned out to be one of the most significant among the peers. In total, MetricsTree supports 61 metrics: 18 on project level, 11 on package level, 22 on class level, 11 on method level. The full list of supported metrics can be found in Table A (Appendix) and on the plugin's GitHub page (https://github.com/b333vv/metricstree).

Along with including a large number of supported metrics, the software design of the plugin is intended for minimizing the amount of rework required to add new types of metrics. All countable Java code elements are available (which the IDE is responsible for), and there are entities for representing metrics. Thus, all that remains to implement a new basic metric is to define an heir in the Visitor hierarchy, describing in it the rules for counting Java code elements of this new metric. If a derivative metric is added, it is necessary to make an heir in the metrics hierarchy, describing in it the formula for complexing the base and derivative metrics to get the value of the new metric.

Table 1 compares MetricsTree with its closest counterparts in terms of the main families of supported metrics (open-source tools are highlighted in gray).

Along with the calculation of metrics values, MetricsTree implements functions for setting reference metrics values. These values were derived based on the analysis of studies of metrics correlation with software attributes. At the same time, to adapt to the peculiarities of development, the values of the benchmark intervals can be changed. To increase the speed of assimilation of information by developers, MetricsTree uses color indication to indicate the extent to which metrics fall within the benchmark interval.

Key features

Trees

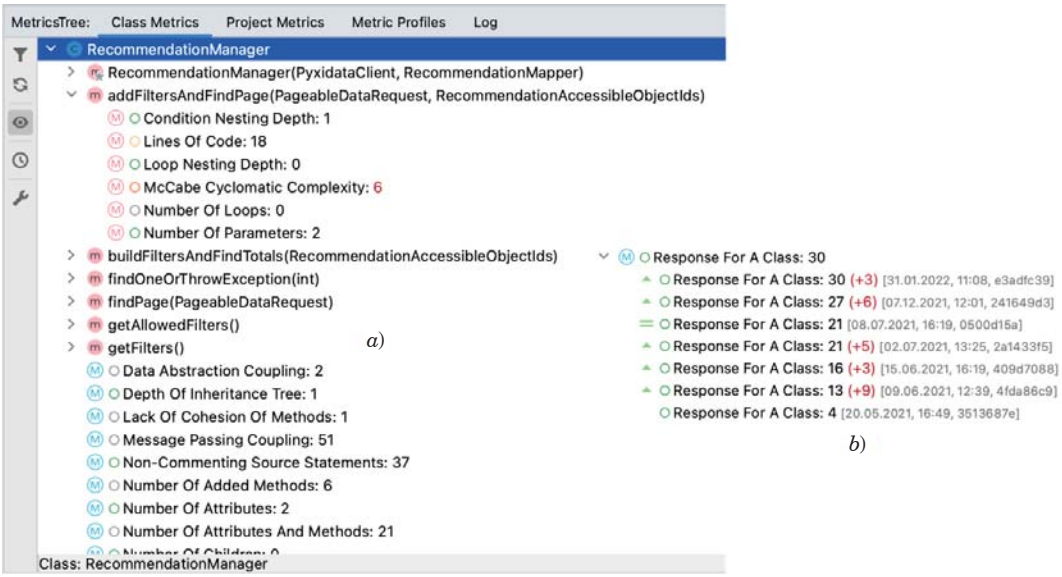
Trees grouping metrics by class and method level are used to display the hierarchy of class metrics (Fig. 1, *a*). The grouping levels for mapping the application metrics are application, package, class, and method (Fig. 2, *a*, *b*). In addition, the project metrics tree groups metrics by metrics authors (e.g., Chidamber — Kemerer and Robert Martin's metrics, etc.). For each metric, a choice of 5 colors is used: green (if the metric value is within the control limits), yellow (if the metric value is slightly out of bounds), red (if the metric value is out of bounds), bright red (if the metric value is significantly out of bounds). The function for getting the evolution of the metric value of each class during development is also implemented (Fig. 1, *b*).

Treemaps

Treemap charts are used to view data in a hierarchical view, using a grid of rectangles with sizes

■ ***Table 1.*** Support for families of metrics by metric analysis tools

App/Metric Set	MOOD [16]	QMOOD [17]	Robert C. Martin [18]	Chidamber – Kemerer [19]	Lorenz – Kidd [20]	Li – Henry [21]	Lanza – Marinescu [22]	Holstead [23]	MI [24]	Statistic
CAST's AIP (doc.castsoftware.com/display/TECHNOS)			+			+				+
CKJM (www.spinellis.gr/sw/ckjm)				+						
CMT++/CMTJava (www.verifysoft.com/en_cmtx.html)								+	+	+
CodeMR (www.codemr.co.uk)			+	+			+			+
Halstead Metrics Tool (sourceforge.net/projects/halsteadmetricstool)								+		
JHawk (www.virtualmachinery.com/jhawkprod.htm)			+	+		+		+	+	
MetricsReloaded (github.com/BasLeijdekkers/Metrics-Reloaded)	+		+	+	+	+		+		+
MetricsTree	+	+	+	+	+	+	+			+
PMD							+			+
SonarQube						+			+	+
Understand (emenda.com/scitools-understand)			+	+		+				+

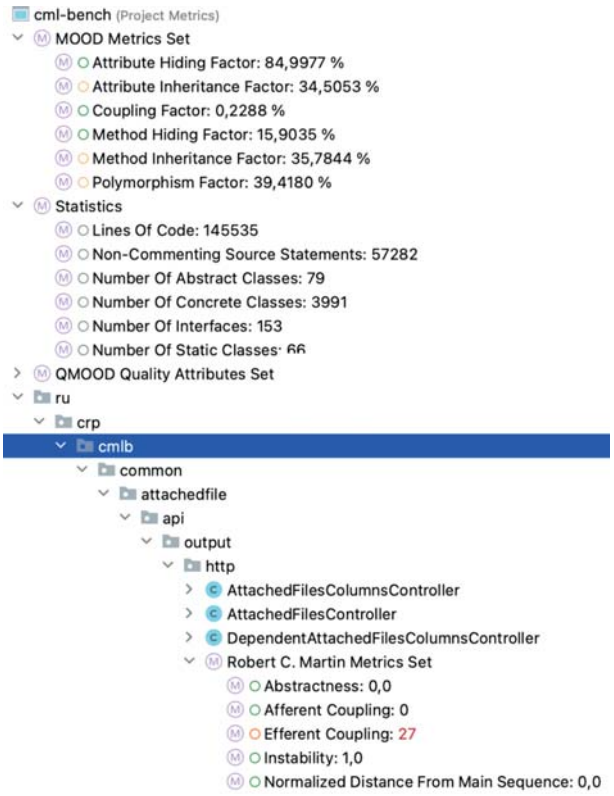


■ ***Fig. 1.*** Class metrics tree (*a*) and metric evolution tree using the RFC metric as an example (*b*)

proportional to the size of the class, which is defined by the Non-Commenting Source Statements metric. The color of each rectangle (5 shades from red to green) reflects the degree to which the value of the metric selected in the list on the left corresponds to the reference range. Clicking on the list in the left part of the window (Fig. 3) displays the Treemap chart in the middle, clicking on the rectangle in the middle representing the class displays all the metrics of this class on the right.

Charts

Several types of charts are used to improve the efficiency of metric analysis of Java source code. Some of them clearly show the general metric picture reflecting the state of source code quality



■ ***Fig. 2.*** Project metrics tree

(Figs. 4–6). Another part visualizes different types of relationships between metric values and metric profiles and serves for more in-depth metric analysis of structural properties of Java code (Figs. 7–9).

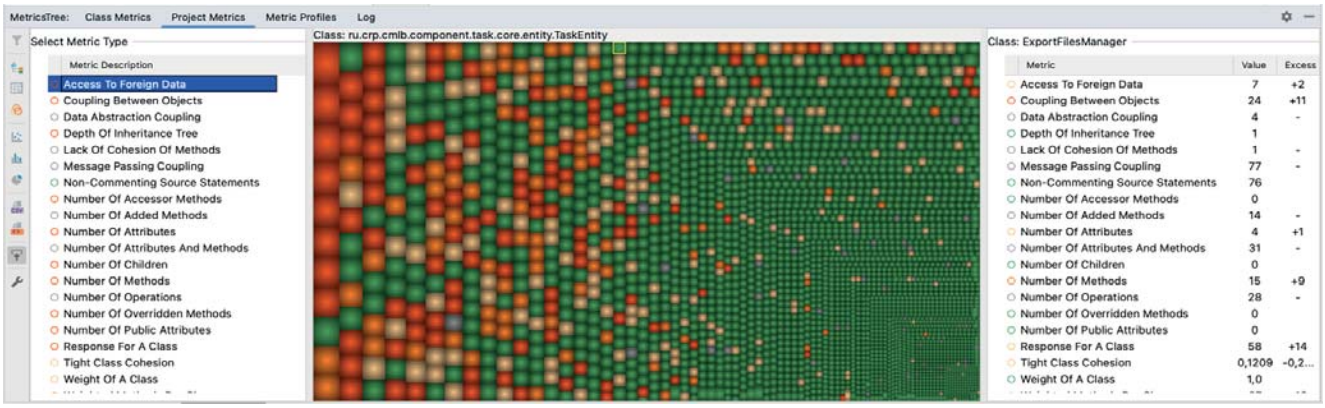
Synergy with the IDE

Once a class is selected in the project tree, its source code is opened in the editor, its metrics are simultaneously calculated and displayed as a tree and a table (Fig. 10, *a*–*d*). The developers do not have to take any additional steps to get the metrics values for the class processed at the moment. Embedding the information about the metrics into the IDE generates a synergy effect, serving to increase the degree of validity and effectiveness of the decisions made by the developers by promptly obtaining the metrics values.

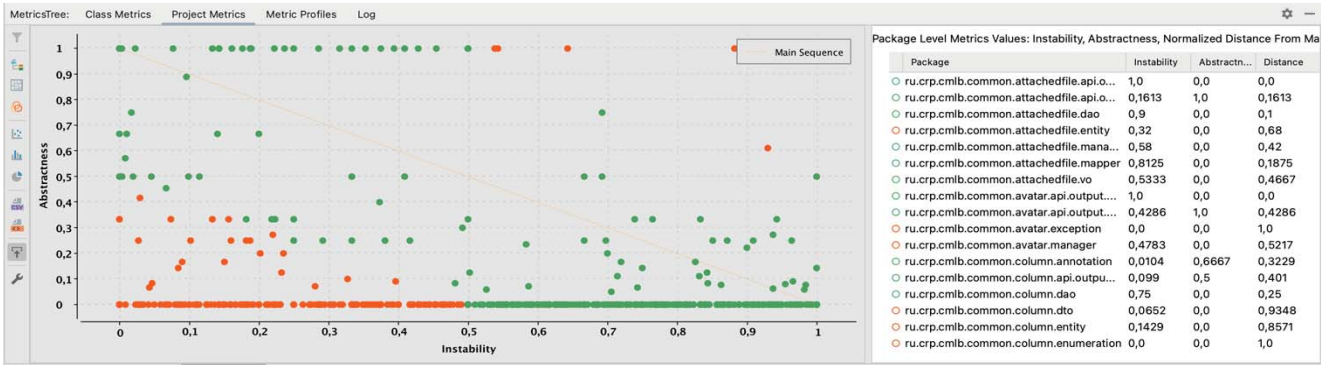
MetricsTree provides an option to control the composition of the metrics displayed in the tree, as well as the reference thresholds for base and derived metrics values.

Metric profiles

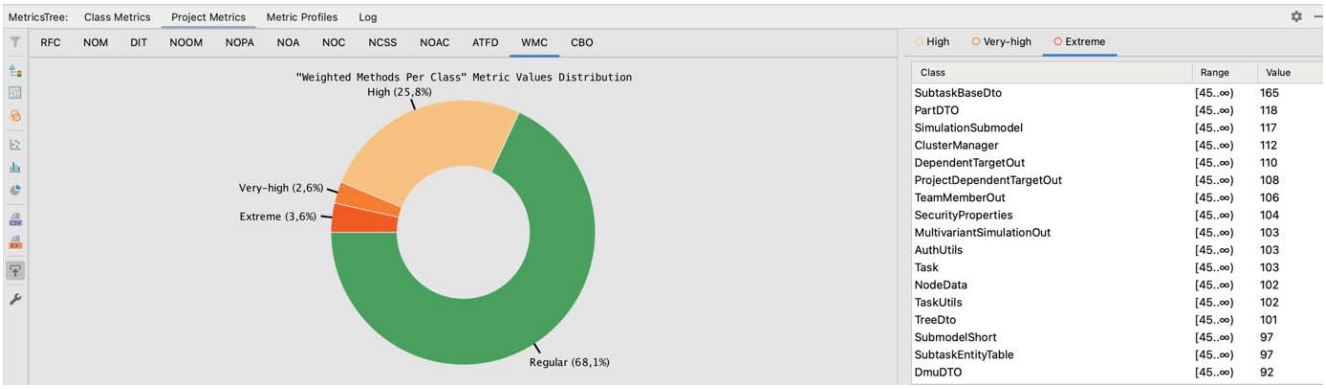
The metric profiles tool was originally inspired by research into the possibility of detecting code smells using metrics [24]. We have extended the interpretation of this tool by calling it the metric



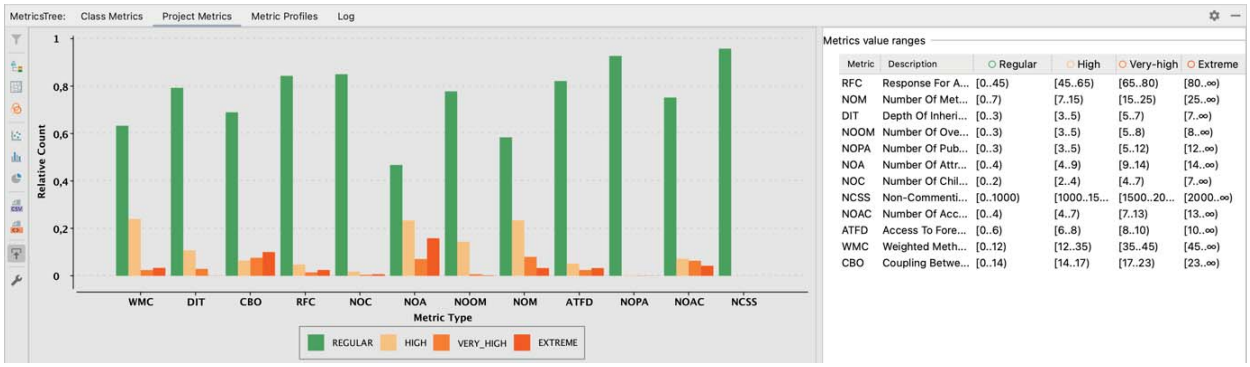
■ ***Fig. 3.*** Treemap with class-level metrics



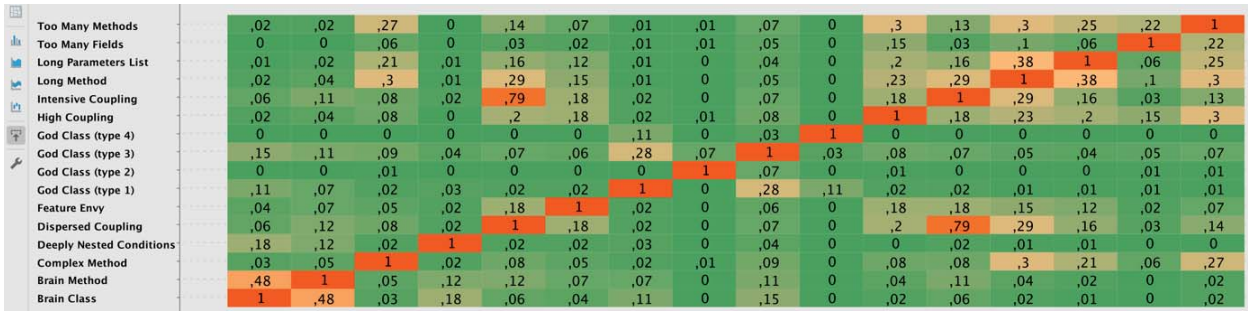
■ ***Fig. 4.*** Abstractness vs. instability distribution



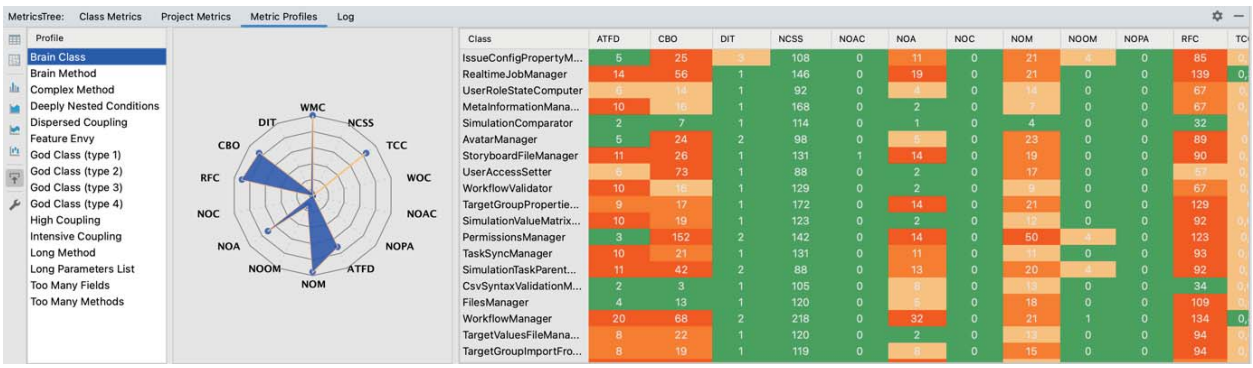
■ ***Fig. 5.*** Pie chart for metrics distribution by type



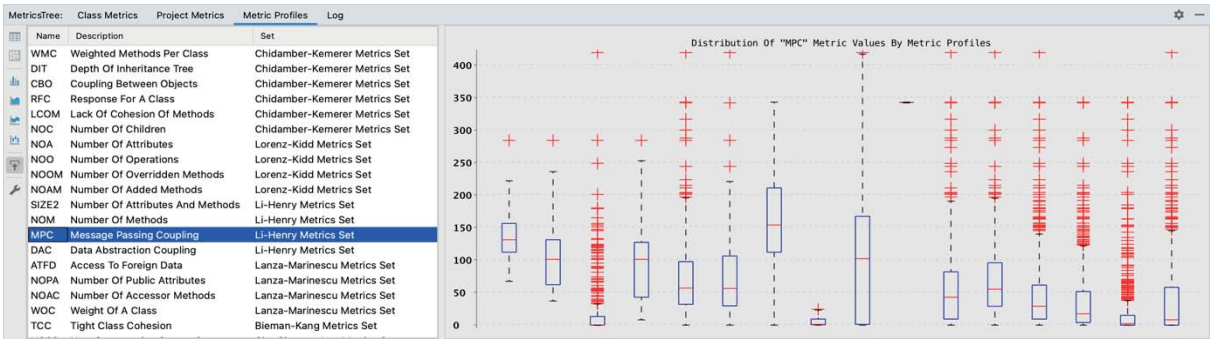
■ ***Fig. 6.*** Metrics distribution



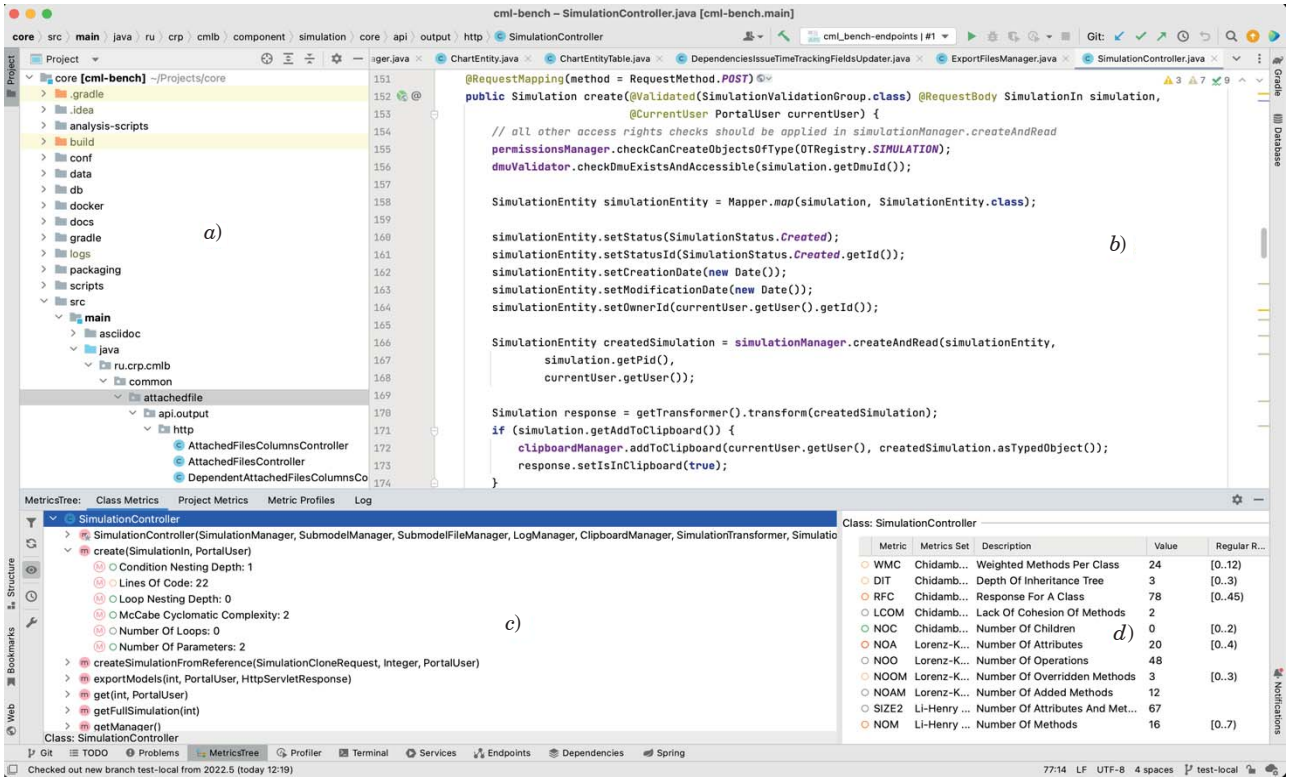
■ ***Fig. 7.*** Correlation between metric profiles



■ ***Fig. 8.*** Distribution of metric values across metric profiles



■ ***Fig. 9.*** Correlation between metric values and metric profiles using the MPC metric as an example



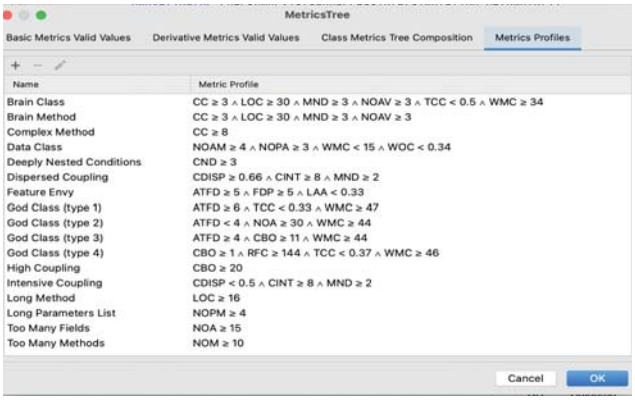
■ ***Fig. 10.*** Main window in IntelliJ IDEA with embedded MetricsTree plugin: *a* — selecting a class in the project tree; *b* — displaying source code in the editor; *c* —results of metrics calculation in the tree; *d* — results of the metrics calculation in the table

profiles formation tool, allowing to identify classes with given sets of metric values by constructing appropriate samples. The conjunct of metric value ranges serves as an analytical specification of the metric profile (Fig. 11).

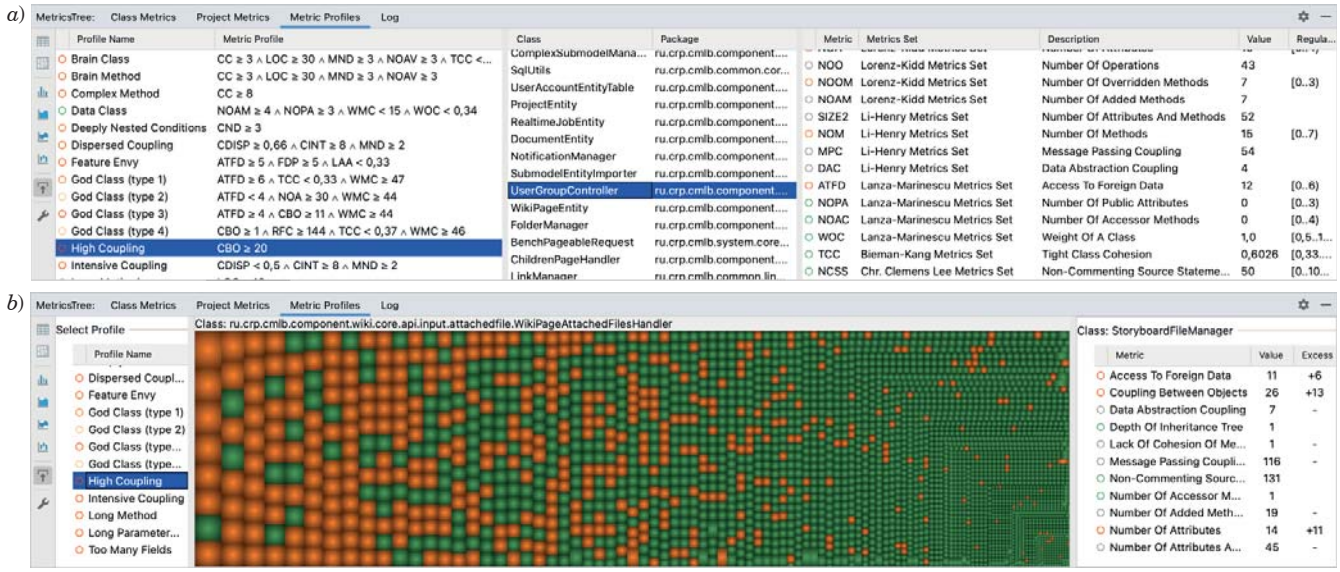
Once defined, metric profiles allow generating subsets of classes with certain metric values, which can be visualized as a table (Fig. 12, *a*) or a Treemap (Fig. 12, *b*).

Comparison with counterparts in terms of functionality

Table 2 presents a comparison of the selected tools for metric analysis within the selected set of



■ ***Fig. 11.*** Dialog box for setting the metric thresholds



■ ***Fig. 12.*** Class distribution by metric profile: *a* — table view; *b* — Treemap view

■ ***Table 2.*** Comparison of metrics calculation tools by functionality

App/Feature	Automatic metric calculation	Metric evolution monitoring	Tabular view	Tree view	Diagram view	Treemap view	Metric profiles	Metric profiles setting	Metric values control	Metric reference value setting
CAST's AIP			+		+				+	
CKJM			+							
CMT++/CMTJava			+						+	
CodeMR			+		+	+	+	+	+	
Halstead Metrics Tool			+							
JHawk			+		+					
MetricsReloaded			+		+				+	
MetricsTree	+	+	+	+	+	+	+	+	+	+
PMD			+				+			
SonarQube			+		+				+	+
Understand			+							

properties. Many of the metrics analysis tools compared have useful features not considered in the presented comparison. The rows with open-source tools are highlighted in gray.

Effects of use

Over the past year, while still using SonarQube and PMD, we have integrated MetricsTree into our quality assurance processes. MetricsTree is implemented as a multifunctional tool, the main aspects of its implementation can be summarized as the following three groups.

Working on the task

The main element of the MetricsTree that accompanies a programmer's daily work in the IDE is the tree of class metrics (see Fig. 10, *c*). It displays the values of metrics at the class and method level, drawing the developer's attention to the class metrics whose values fall outside the control limits by means of color-coding. There are five color codes, which change depending on the degree of deviation from

the reference values. The developer can use the user interface to control the thresholds of the metric value ranges. To help the developer focus on the important tasks, the UI has a feature to control the composition of the metrics in the tree. Developers do not need to take any steps to manage the metric values, as normal work with Java classes in IDEA in the project tree and editor is accompanied by quick calculation and display of metrics values for the class considered at the moment. Based on feedback from our developers, we can conclude that adding real-time metrics information to the typical IDE window structure helps them make the right decisions when working on a certain class or its method in a timelier manner.

Final checks on the task

After completing the task in the local environment, the developer performs a quality check of the new or modified functionality before implementing the changes in the main branch of code. The developer uses the metric profiles tool to look for classes with code smells or defects. If the developer discovers classes for which they are responsible in the generated sample, they fix the defects found before putting the results of the work into the common code base.

Prerelease activities

After the set of tasks making up the new CML-Bench release is completed, several specialized measures are taken as part of our quality assurance regulations.

First, analysis of code smells or defects is performed based on metrics profiles, similar to the one described in the previous section, but project-wide.

The second important component is the analysis of project quality using diagrams. We evaluate the evolution of QMOOD metrics on a diagram, controlling for possible quality degradation associated with refinements. Other diagrams are used to investigate different aspects of the project metrics state at a deeper level: distribution of metrics values, dependencies between different metrics, dependencies between metrics and metrics profiles, etc.

Third, we use an unloading of all project metrics as an XML file and compare it to a similar unloading associated with the previous releases. Assessing the evolution of metrics values, we look for system elements whose quality has deteriorated due to changes made when the current release was prepared, analyze the reasons for the deterioration, and determine ways to fix them.

As noted above, after we developed and started to use MetricsTree, we continued to use SonarQube and PMD. In the year that MetricsTree was adopted to support quality assurance processes, our three CML-Bench development teams created 5 new services and refined 11 existing services, while writing about 72,000 lines of new and modified Java source code. At the same time, the number of code defects identified by SonarQube and PMD during this period decreased by 34%. The improvement in quality metrics from release to release of CML-Bench reinforces our confidence in the effectiveness of automating quality assurance processes with our MetricsTree tool and justifies the importance of its use in the development process.

Conclusion

The plugin we developed for IDE IntelliJ IDEA is designed for metric analysis of structural properties of Java code and has a number of key features:

— is based on rigorous mathematical models for quality and measurement of software quality control theory;

— aims to minimize the cost of adding new types of metrics;

— synergistically extends the space generated by the IDE to quickly add information about quantitative properties of the software entities developed;

— supports different ways to visualize metrics information, i.e., trees, Treemaps, charts of different types to enhance analysis of quantitative properties of Java code;

— implements a mechanism for describing metric profiles and forming class subsets with appropriate combinations of metric values.

MetricsTree has been successfully used for a year to ensure the high quality of the advanced CML-Bench platform for developing and implementing digital twins we are working on.

Our immediate plans include:

— conducting an in-depth study to validate the feasibility of the metric profile mechanism;

— conducting a comprehensive study of the impact of various MetricsTree features used on software quality assurance processes;

— applying MetricsTree to multiple open-source Java projects to increase the generality and validity of the conclusions;

— evolution towards taking into account the other components of software development.

As short-term directions for MetricsTree, we will focus on engineering a tool for formulating analytical expressions for metric profiles. Furthermore, we plan to implement a number of additional metrics:

— Halstead Metrics [23];

— Maintainability Index (MI) [24];

— Decoupling Level (DL) [25];

— Propagation Cost (PC) [26].

In addition, the MetricsTree repository on Github currently contains about 30 Issues created by plugin users from different countries; we intend to analyze the feasibility of implementing these in the near future.

Appendix

■ ***Table A.*** List of metrics calculated by MetricsTree

Level	Family (Author)	Abbreviation	Description	Source
Project	Chr. Clemens Lee	NCSS	Non-Commenting Source Statements	www2.informatik.hu-berlin.de/swt/intkoop/jcse/tools/JavaNCSS%20-%20A%20Source%20Measurement%20Suite%20for%20Java.html
Package	Chr. Clemens Lee	NCSS	Non-Commenting Source Statements	
Class	Chr. Clemens Lee	NCSS	Non-Commenting Source Statements	
Project	–	LOC	Lines of Code	–
Package	–	LOC	Lines of Code	–
Class	–	LOC	Lines of Code	–
Method	–	LOC	Lines of Code	–
Project	–	NOC	Number of Concrete Classes	–
Package	–	NOC	Number of Concrete Classes	–
Project	–	NOA	Number of Abstract Classes	–
Package	–	NOA	Number of Abstract Classes	–
Project	–	NOSC	Number of Static Classes	–
Package	–	NOSC	Number of Static Classes	–
Project	–	NOI	Number of Interfaces	–
Package	–	NOI	Number of Interfaces	–
Project	MOOD	MHF	Method Hiding Factor	[16]
Project	MOOD	AHF	Attribute Hiding Factor	
Project	MOOD	MIF	Method Inheritance Factor	
Project	MOOD	AIF	Attribute Inheritance Factor	
Project	MOOD	PF	Polymorphism Factor	
Project	MOOD	CF	Coupling Factor	
Project	QMOOD	–	Reusability	[17]
Project	QMOOD	–	Flexibility	
Project	QMOOD	–	Understandability	
Project	QMOOD	–	Functionality	
Project	QMOOD	–	Extendibility	
Project	QMOOD	–	Effectiveness	
Package	Robert C. Martin	Ce	Efferent Coupling	[18]
Package	Robert C. Martin	Ca	Afferent Coupling	
Package	Robert C. Martin	I	Instability	
Package	Robert C. Martin	A	Abstractness	
Package	Robert C. Martin	D	Normalized Distance from Main Sequence	
Class	Chidamber — Kemerer	WMC	Weighted methods per class	[19]
Class	Chidamber — Kemerer	DIT	Depth of Inheritance Tree	
Class	Chidamber — Kemerer	NOC	Number of Children	
Class	Chidamber — Kemerer	CBO	Coupling between object classes	

■ *Ending of Table A*

Level	Family (Author)	Abbreviation	Description	Source
Class	Chidamber — Kemerer	RFC	Response for a Class	
Class	Chidamber — Kemerer	LCOM	Lack of cohesion in methods	
Class	Lorenz – Kidd	NOA	Number of Attributes	[20]
Class	Lorenz – Kidd	NOO	Number of Operations	
Class	Lorenz – Kidd	NOAM	Number of Added Methods	
Class	Lorenz – Kidd	NOOM	Number of Overridden Methods	
Class	Li – Henry	SIZE2	Number of Attributes and Methods	[21]
Class	Li – Henry	MPC	Message Passing Coupling	
Class	Li – Henry	DAC	Data Abstraction Coupling	
Class	Li – Henry	NOM	Number of Methods	
Class	Lanza – Marinescu	ATFD	Access to Foreign Data	[22]
Class	Lanza – Marinescu	NOPA	Number of Public Attributes	
Class	Lanza – Marinescu	–	Number of Accessor Methods	
Class	Lanza – Marinescu	WOC	Weight of a Class	
Class	Bieman – Kang	TCC	Tight Class Cohesion	[27]
Method	McCabe	CC	McCabe Cyclomatic Complexity	–
Method	–	–	Maximum Nesting Depth	–
Method	–	–	Loop Nesting Depth	–
Method	–	–	Condition Nesting Depth	–
Method	–	–	Number of Loops	–
Method	–	LAA	Locality of Attribute Accesses	–
Method	–	FDP	Foreign Data Providers	–
Method	–	NOAV	Number of Accessed Variables	–
Method	–	CINT	Coupling Intensity	–
Method	–	CDISP	Coupling Dispersion	–

References

1. State Standard R 57700.37-2021. *Digital Twins of Products. General Provisions*. Moscow, Rossijskij institut standartizacii Publ., 2021. 11 p. (In Russian).
2. Borovkov A. I., Rozhdestvenskiy O. I., Pavlova E., Glazunov A. I., and Savichev K. Key barriers of digital transformation of the high-technology manufacturing: An evaluation method. *Sustainability*, 2021, vol. 13, no. 20: 11153. doi:10.3390/su132011153
3. Borovkov A. I., Gamzikova A. A., Kukushkin K. V., Ryabov Yu. A. *Cifrovye dvojniki v vysokotekhnologichnoj promyshlennosti* [Digital Twins in the High-Technology Manufacturing Industry]. Politekhnicheskij universitet Publ., Saint-Petersburg, 2019. 62 p. (In Russian).
4. Burakov V. V. *Upravlenie kachestvom programmny`kh sredstv* [Software Quality Management]. Saint-Petersburg, GUAP Publ., 2009. 287 p. (In Russian).
5. Burakov V. V. The use of simulation modeling to improve software quality. *Trudy devyatoj vserossijskoj nauchno-prakticheskoj konferencii po imitacionnomu modelirovaniyu i ego primeneniyu v nauke i promyshlennosti "Imitacionnoe modelirovanie. Teoriya i praktika" IMMOD-2019* [Proc. of the IX All-Russian Scientific and Practical Conf. on Simulation Modelling and its Applications in Science and Industry. "Simulation Modelling. Theory and Practice" (IMMOD-2019)], 2019, pp. 368–374 (In Russian).
6. Burakov V. V., Kulakov A. Yu., Cherniy A. N. Software maintenance evaluation. *Informatization and Communication*, 2019, no. 4, pp. 14–21 (In Russian). doi:10.34219/2078-8320-2019-10-4-14-21
7. Kumar N. R., Viji C., and Duraisamy S. Measuring cohesion and coupling in object oriented system using Java reflection. *ARPN Journal of Engineering and Applied Sciences*, 2015, vol. 10, no. 7, pp. 3096–3101.

8. Ankush Vesra, Rahul. A study of various static and dynamic metrics for open source software. *International Journal of Computer Applications*, 2015, vol. 122, no. 10, pp. 17–19. doi:10.5120/21736-4927

9. Nicolaescu A., Lichter H., Xu Yi. Evolution of object oriented coupling metrics: A sampling of 25 years of research. *Software Architecture and Metrics (SAM)*, 2015 IEEE/ACM 2nd Intern. Workshop, May 2015. doi:10.1109/SAM 2015.14

10. Schnoor H., Hasselbring W. Toward measuring software coupling via weighted dynamic metrics. *40th Intern. Conf. on Software Engineering: Common Proc.*, ACM/IEEE, 2018, pp. 342–343. doi:10.1145/3183440.3195000

11. Tkachuk M., Nagornyi K., and Gamzayer R. Models, methods and tools for effectiveness estimation of post object oriented technologies in software maintenance. *ICTERI 2015*, CCIS 594, Springer International Publisher Switzerland, 2016, pp. 20–37.

12. Padhy N., Satapathy S., and Singh R. P. State-of-the-art object oriented metrics and its reusability: A decade review. *Smart Computing and Informatics*, 2018, pp. 431–441. doi:10.1007/978-981-10-5544-7_42

13. Padhy N., Satapathy S., Singh R. P., and Sethlani J. A systematic literature review of an object oriented metrics components: Case study for evaluation of reusability criteria. *Intern. Conf. on Advanced Studies in Engineering and Sciences*, 2017, pp. 49–61.

14. Mao C., Xu Ch. Entropy based dynamic complexity metrics for service oriented systems. *24th Asia-Pacific Software Engineering Conf. Workshops*, IEEE, 2017, pp. 90–97. doi:10.1109/APSECW.2017.14

15. Aswini S., Yazhini M. An assessment framework of routing complexities using LOC metrics. *Intern. Conf. on Innovations in Power and Advanced Computing Technologies*, IEEE, 2017, pp. 1–6. doi:10.1109/IPACT.2017.8245022

16. Brito e Abreu F. and Carapuça R. Object-oriented software engineering: Measuring and controlling the development process. *4th Intern. Conf. on Software Quality*, Mc Lean, VA, USA, 1994, pp. 1–8.

17. Bansiya Ja., and Davis C. G. A hierarchical model for object-oriented design quality assessment. *IEEE Transactions on Software Engineering*, 2002, vol. 28, iss. 1, pp. 4–17.

18. Martin R. C. *Agile Software Development: Principles, Patterns, and Practices*. Alant Apt Series. Prentice Hall, Upper Saddle River, NJ, USA, 2002. 552 p.

19. Chidamber S. R., and Kemerer C. F. A metrics suite for object oriented design. *IEEE Transactions on Software Engineering*, 1994, vol. 20 (6), pp. 476–493.

20. Lorenz M., Kidd J. *Object Oriented Software Metrics*. Pearson, 2008. 160 p.

21. Li W., and Henry S. Object-oriented metrics that predict maintainability. *Journal of Systems and Software*, 1993, vol. 23, iss. 2, pp. 111–122.

22. Lanza M., Marinescu R. *Object-oriented metrics in practice*: *Using software metrics to characterize, evaluate, and improve the design of object-oriented systems*. Springer, 2006. 207 p. doi:10.1007/3-540-39538-5

23. Chhillar R. S., and Gahlot S. An evolution of software metrics: A review. *Proc. of the Intern. Conf. on Advances in Image Processing, ICAIP 2017*, New York, NY, USA, 2017, pp. 139–143. doi:10.1145/3133264.3133297

24. Coleman D., Ash D., Lowther B., and Oman P. Using metrics to evaluate software system maintainability. *Computer*, 1994, vol. 27, no. 8, pp. 44–49. doi:10.1109/2.303623

25. Mo R., Cai Ya., Kazman R., Xiao Lu, Feng Q. Decoupling level: A new metric for architectural maintenance complexity. *Proc. of the Intern. Conf. on Software Engineering*, Austin, TX, 2016, pp. 499–510.

26. MacCormack A., Rusnak J., Baldwin C. Y. Exploring the structure of complex software designs: An empirical study of open source and proprietary code. *Management Science*, 2006, vol. 52, iss. 7, pp. 1015–1030.

27. Bieman J. M., Kang B. Cohesion and reuse in an object-oriented system. *Proc. of the 1995 Symp. on Software Reusability*, Seattle, Washington, United States, 1995, pp. 259–262.

УДК 004.05
doi:10.31799/1684-8853-2023-1-17-28
EDN: ORVZMP

Средство для углубленного метрического анализа исходного кода на Java

В. В. Бураков[а], доктор техн. наук, доцент, orcid.org/0000-0002-0158-8681, burakov@compmechlab.com
А. И. Боровков[б], канд. техн. наук, профессор, orcid.org/0000-0003-3177-0959
[а]ООО Лаборатория «Вычислительная механика», Гжатская ул., 21, к. 2а, Санкт-Петербург, 195220, РФ
[б]Санкт-Петербургский политехнический университет Петра Великого, Политехническая ул., 29, Санкт-Петербург, 195251, РФ

Введение: несмотря на значительные усилия многочисленных исследователей и разработчиков, процедуры оценки качества программного обеспечения все еще нуждаются в формализации и автоматизации. **Цель:** разработать специализированное программное средство, предназначенное для количественной оценки структурных свойств Java-кода. **Результаты:** разработано программное средство MetricsTree, которое рассчитывает 61 устоявшуюся объектно-ориентированную метрику (это один из самых больших наборов среди аналогичных инструментов). MetricsTree интегрировано в IDE для обеспечения максимально быстрой доставки инфор-

мации, содержит уникальные средства визуализации для повышения эффективности анализа метрик, а также реализует механизм профилей метрик для выбора классов на основе набора значений метрик. **Практическая значимость:** в результате применения MetricsTree для автоматизации процессов обеспечения качества при разработке флагманской системы Санкт-Петербургского государственного политехнического университета Петра Великого CML-Bench (платформы для разработки и применения цифровых двойников) в течение года среднее количество выявленных программных дефектов внешними средствами сократилось на 34 %.

Ключевые слова — качество программного обеспечения, метрики программного обеспечения, объектно-ориентированные метрики, метрики исходного кода на Java, программные средства расчета метрики, метрический анализ, визуализация программного обеспечения, дефекты исходного кода.

Для цитирования: Burakov V. V., Borovkov A. I. Advanced metric analysis tool for Java source code. *Информационно-управляющие системы*, 2023, № 1, с. 17–28. doi:10.31799/1684-8853-2023-1-17-28, EDN: ORVZMP

For citation: Burakov V. V., Borovkov A. I. Advanced metric analysis tool for Java source code. *Informatsionno-upravliaiushchie sistemy* [Information and Control Systems], 2023, no. 1, pp. 17–28. doi:10.31799/1684-8853-2023-1-17-28, EDN: ORVZMP

УВАЖАЕМЫЕ АВТОРЫ!

Научные базы данных, включая Scopus и Web of Science, обрабатывают данные автоматически. С одной стороны, это ускоряет процесс обработки данных, с другой — различия в транслитерации ФИО, неточные данные о месте работы, области научного знания и т. д. приводят к тому, что в базах оказывается несколько авторских страниц для одного и того же человека. В результате для всех по отдельности считаются индексы цитирования, что снижает рейтинг ученого.

Для идентификации авторов в сетях Thomson Reuters проводит регистрацию с присвоением уникального индекса (ID) для каждого из авторов научных публикаций.

Процедура получения ID бесплатна и очень проста, есть возможность провести регистрацию на 12 языках, включая русский (чтобы выбрать язык, кликните на зеленое поле вверху справа на стартовой странице): https://orcid.org