Intelligent System for Determining the Sufficiency of Metric Information in the Software Requirements Specifications

Tetiana Hovorushchenko [1][0000-0002-7942-1857] and Olga Pavlova [2][0000-0003-2905-0215]

Khmelnitsky National University, Institutska str., 11, Khmelnitsky, 29016, Ukraine
[1] tat_yana@ukr.net
[2] olya1607pavlova@gmail.com

Abstract. The paper is devoted to the developing the intelligent system for determining the sufficiency of metric information in the software requirements specifications (SRS), which provides on the basis of the natural language processing of SRS: conclusion about the sufficiency of metric information (indicators for metrics calculation) in SRS, numerical assessment of the sufficiency level of metric information in the SRS, visualization of missing indicators for metrics calculation. The developed intelligent system provides an increase in the sufficiency of information by 12.71-50.28% for the SRS of the information and analytical system for the accounting of therapeutic and diagnostic activities provided to the wounded during transportation. In general, the developed system provides an increase in the sufficiency of metric information in the SRS to 100% – if it's necessary (for critical software) or at the customer's request. The developed intelligent system for determining the sufficiency of metric information in the SRS can be used during the software development for government agencies, military formations and law enforcement agencies, commercial organizations (both for organizations-developers of software and for organizations-customers of software).

Keywords: Software Metrics, Software Requirements Specification (SRS), Sufficiency of Metric Information, Intelligent Agent (IA), Intelligent System (IS).

1 Introduction

Given the fact that in today's world software development has become one of the most expensive industries, and any bottlenecks in the technological process of its creation can lead to unwanted results, one of the main requirements of users to modern software is its high quality and low complexity. According to ISO 25010 [1], ISO 25030 [2], ISO 19759 (SWEBOK) [3], the quality of software is the ability of software to meet the claimed and predictable customer's needs during its use under certain conditions. Software quality in ISO 9000 [4] and ISO 9001 [5] is the degree of accordance of software to the requirements. Definition of quality from standards [4, 5] doesn't take into account the fact that requirements may not fully reflect the customers' needs, then meeting the requirements isn't meeting the customers' needs,

so such software cannot be considered qualitative (in fact, there will only be formal quality satisfaction). The structural complexity of software is the number of interacting components, the number of links between components and the complexity of their interaction [6].

Today, there are a number of models that provide the calculation of the software quality and complexity, but the multiplicity of interpretation of these characteristics complicates such calculations. Most models are based on the use of different software metrics. According to ISO 24765 [7], the software metric is a measure, which gives the numeric value of a certain software feature as a weighted arithmetic mean taking into account the values of the indicators of this metric and their weights.

The using of quantitative metrics had to help in solving a number of practical tasks [8]: 1) predicting the number of bugs in the software from the beginning of the project; 2) predicting the level of software quality, software complexity and its maintenance on the basis of analysis of the results of the design stage; 3) predicting the level of complexity and quality of the testing processes and the number of unidentified bugs based on analysis of the source code; 4) predicting the final size of the source code based on the analysis of the complexity assessments of the design stage; 5) determining the impact of certain features of source code on the quality of the ready software; 6) control of project development stages; 7) analysis of explicit and hidden defects; 8) identifying the best methods and technologies of development on the basis of experimental comparison.

The modern software industry has accumulated a large number of metrics, that assess the certain production and operating features of software, but modern software is not ideal in terms of its quality [9], and in the field of evaluation and prediction of software characteristics based on the analysis of metrics remains a series of unsolved issues [8, 10]: 1) there are no common standards for metrics, which leads to a subjective choice of metrics – more than a thousand metrics has created, each developer of the "measuring" system offers its own methods and metrics for assessing the quality; 2) there is a problem of the complexity and subjectivity of the interpretation of the values of metrics – the value of metrics, which were obtained using "measuring" systems, are non-informative or little-informative for the user, for the customer, and often for the programmer; 3) tools of automating the calculation of metrics are aimed at metrology of the finished source code, and aren't aimed at prediction and calculation of software metrics at the design stage, when there is no source code, and there are only informational, functional and behavioural models of requirements analysis; 4) there is a problem of low level of automation of analysis and processing of software metrics – only the processes of collecting, registering and calculating the metric information are currently automated; 5) all known automated tools are aimed, in general, at quality assessment, but none of them is aimed at ensuring the adequate level of quality.

The complexity of the justification of the choice and interpretation of metrics during making the production decisions, and ignoring the stages of the software lifecycle don't allow the full use of metrics for evaluating and predicting the software characteristics at the early stages of the software lifecycle. At present, *the actual task* is calculating the metrics' values for estimating and predicting the quality and complexity of software at the early stages of the lifecycle.

2 State-of-the-Art

Let's consider the known methods and tools for evaluating the values of complexity and quality metrics.

The German National Research Center of Information Technology (GMD) has developed the ProcePT project [11], which includes information models of software development processes, and quality assessment methods based on the metrics, in particular, the SMV component aims at measuring software metrics and quantifying the software characteristics ; the SPM component aims to assess the quality of development processes based on quantitative indicators. These products are designed to evaluate the finished source code.

IBM corporation offers the methodology for creating complex software systems, called Cleanroom Software Engineering [12]. The Cleanroom tool for automated testing and evaluation of software reliability is the Cleanroom Certification Assistant environment, that uses statistical results of testing to calculate software reliability metrics by mathematical methods.

The Logiscope package [13] is a set of software tools (TestChecker, RuleChecker, Audit) that conduct comprehensive software testing and improve its quality. The basis of the package is the idea of analyzing the source code. The Logiscope package is designed to qualitatively evaluations of source code and identification of code's fragments, where the occurrence of bugs is most likely. After analyzing the code, Logiscope generates a set of various metric information in the form of quantitative metrics (over 200 types of metrics) about the code, its positive and negative features, generates a complete report that provides the conclusions about the quality of the code.

Pure Software company, a leading developer of automated software tools for creating high-quality software, offers the Purify system [14], which provides to identify a variety of software bugs and to calculate metrics of software complexity. The Purify system is mainly oriented on source code, rarely it can work without code, but it is not designed to work with the requirements and the software requirements specifications (SRS).

The Hindsight tool of IntegriSoft [15] analyzes the source code, measures the source code and calculates the values of software product metrics for their use in quality assessment. The cyclomantic complexity, data complexity, Halstead's metrics, complexity of software architecture are calculated.

The EzCover tool [15] measures the software and calculates the following metrics: cyclomatic complexity, modified complexity, data complexity, number of empty lines, number of commented lines, number of executable lines.

Metricate tool [15] probes practically all aspects of software companies activity: efficiency of technological processes, quality of source code, level of project management, cost of execution of different stages, productivity of the develooped system, productivity of developers' work and quality of finished products.

DMS tool [16] provides a calculation of a number of metrics based on the analysis of the source code. CAST Application Intelligence Platform by CAST [17] provides detailed, audience-specific dashboards to measure quality and productivity. ConQAT [18] provides the continuous quality assessment toolkit that allows flexible

configuration of quality analyses and dashboards. GrammaTech CodeSonar [19] calculates the software metrics for C, C++, Objective-C, and Java source code. Moose [20] is the software analysis platform with many tools to manipulate, assess or visualize software; it can evolve to a more generic data analysis platform. Parasoft [21] provides static analysis (pattern-based, flow-based, in-line, metrics) for C, C++, Java, .NET. SideCI [22] is the static code analysis based automated code review tool; checks style, quality, dependencies, security and bugs. Sonargraph [23] provides the dependency analysis, automated architecture check, metrics and the ability to add custom metrics and code-checkers. SonarQube [24] provides tracks code complexity, unit test coverage and duplication. CppCheck [25] is the open-source tool for analysis of the source code and calculation of the software complexity metrics.

In the paper [26] the tool is developed, named as SWMetrics, using Microsoft Visual Studio C# to compute a metrics of LOC, SLOC and complexity based the cyclomatic complexity metric for quality measurement for many format languages of the source of code.

The paper [27] discusses how the Expert System can be used to automate the selection and implementation process of software quality assurance and provides recommendations for building an expert system for software quality assurance which can be used to help software-producing organizations in selecting the most suitable models to be adopted according to their properties and needs.

Authors of the paper [28] present a software quality support tool, a Java source code evaluator and a code profiler based on computational intelligence techniques that represent a new approach to evaluate and identify inaccurate source code usage and transitively, the software product itself. The aim of this project is to provide the software development industry with a new tool to increase software quality by extending the value of source code metrics through computational intelligence.

Authors of [29] discuss software measurement and metrics and their fundamental role in the software development life cycle, with focus on software test metrics, discuss their key role in software testing process and also classify and systematically analyze the various test metrics.

The proposed in [30] BornBaby model is a whole new dimension of Artificial Intelligence for software engineering domain: attempts are being made to come up with a software synthesis program, which is able to write programs on its own, like human programmers; defines how a program must learn in order to come up with solutions; emphasizes on how the program must learn the data based on natural concepts of living beings and the implementation is generic.

A fuzzy logic reputable paradigm is proposed in [31] for predicting software defect density on individual phases of the software development lifecycle.

The thesis [32] presents methodological investigations using search-based techniques, which are relevant to the task of software quality measurement and prediction, using search-based techniques in large-scale projects during verification, validation and testing of software.

The general disadvantages of the above tools of quality assessment are: a subjective dependence of the choice of metrics that tools of automating the calculation of metric information will calculate; the subjectivity of metric

interpretation, because the exact (standard) values of the metrics are absent; the focus of automated "measuring" tools is the testing and metrology of the finished source code, but not the prediction and calculation of software metrics at the design stage.

In [8] the neuronet method for software quality evaluation and prediction is proposed. The authors of [8] selected 24 metrics of software quality and complexity, which can be calculated already at the design stage (with exact or predicted values). The values of such metrics are analyzed by the artificial neural network, which, based on this analysis, provides assessments of the complexity and quality of the software project, and the predicted assessments of the complexity and quality of the future software. The obtained assessments are processed according to the developed production rules (which were formed on the basis of the empirically obtained threshold values), and as a result the user gets conclusions on the level of complexity and quality of the software project, and the conclusions-predictions on the level of complexity and quality of the future software. The developed method focuses not on the source code, but on the SRS, but is based on the analysis of the ready values of 24 metrics of complexity and quality (which depend on 72 indicators, including 42 different indicators) and doesn't take into account the possibility or impossibility of calculation such metrics on the basis of the information of the SRS.

Then, *the task* of assessing the sufficiency of metric information at the early stages of the software lifecycle, in particular, in the SRS (as the possibility of obtaining the indicators for calculation of metrics' values), *is actual*. Although completeness of software requirements is desirable, determination of completeness of the set of requirements is not realistic as was proved in [33]. The insufficiency of information on indicators in requirements will lead, accordingly, to the impossibility of calculating the values of certain metrics and hence to the reduction of the validity of assessments of software quality and complexity at the early stages of the lifecycle. Today the assessment of sufficiency of software safety requirements is known [33, 34].

Then, *the aim of this study* is the developments of the intelligent system for determining the sufficiency of metric information in the SRS, which, based on the natural language processing of SRS, will provide conclusions about the sufficiency of metric information in the SRS (of indicators for calculation of the chosen in [8] metrics).

3 Intelligent System for Determining the Sufficiency of Metric Information in the Software Requirements Specifications

Intelligent system (IS) for determining the sufficiency of metric information in the SRS is developed as an agent-oriented system, which consists of two intelligent agents (Fig. 1): the intelligent agent (IA) for parsing the SRS on the search of metric information (indicators for calculation of metrics) and the IA for determining the sufficiency of metric information in the SRS.

Both intelligent agents are built on the basis of the ontological approach. IAs use the early developed base ontology of the subject domain "Software Engineering" (part "Quality and complexity of software. Metric analysis") as the known knowledge.

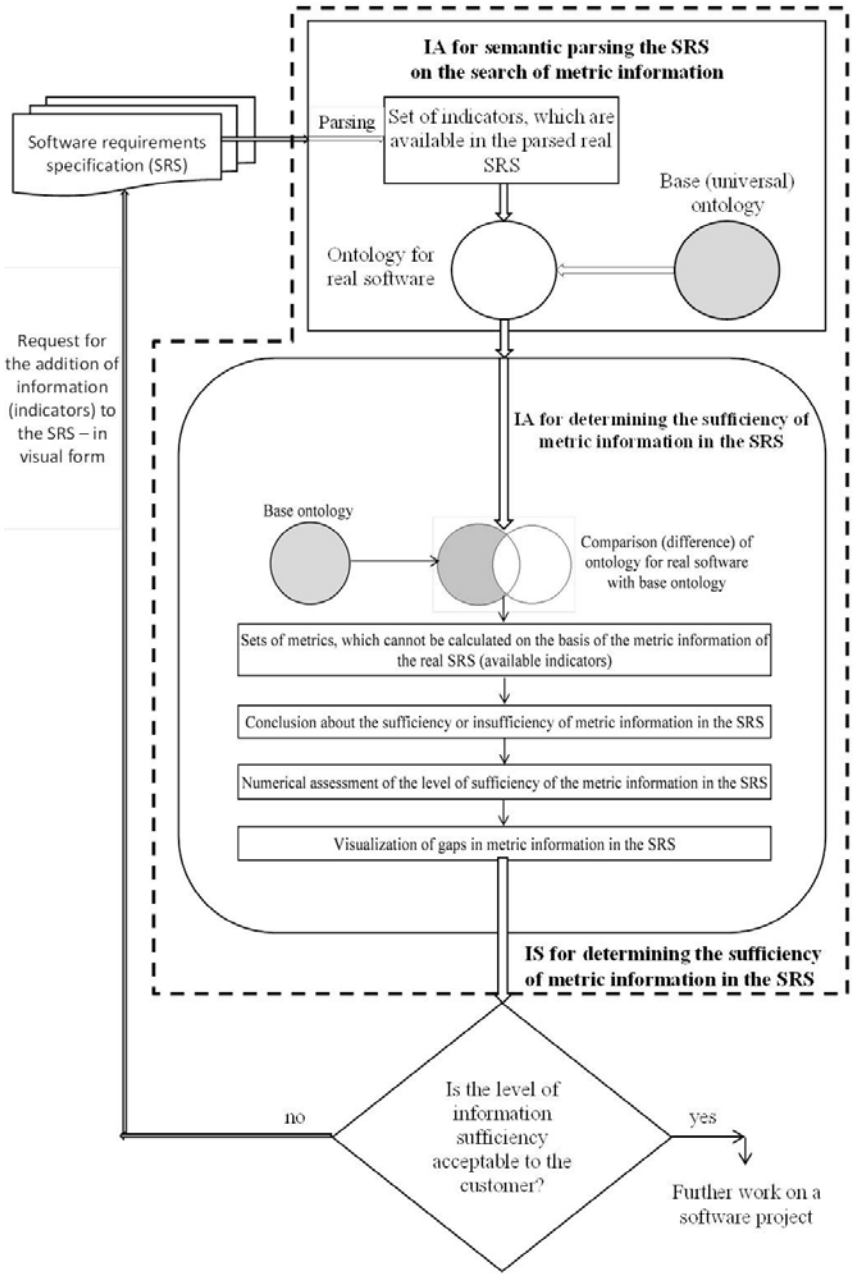

Fig. 1. Intelligent system for determining the sufficiency of metric information in the SRS.

The method of activity of the IA for semantic parsing the SRS on the search of metric information (indicators for the calculation of metrics) consists of the following steps:

1. Search of each indicator of the base ontology "Software Engineering" (part "Quality and complexity of software. Metric analysis") in the SRS for the real software (such the base ontology is contained in the agent's knowledge base).
2. If <indicator$_i$> is in the SRS, then <indicator$_i$> belongs to the set of available indicators, i=1..42 (since, according to [8], there are 42 different indicators, that effects on metrics of quality and complexity of software).
3. If <indicator$_i$> is not in the SRS, then <indicator$_i$> belongs to the set of missing indicators, i=1..42.
4. All indicators of the set of missing indicators are deleted from the base ontology "Software Engineering" (part "Quality and complexity of software. Metric analysis").
5. Checking that all indicators of the set of available indicators are in the ontology after its modification in the previous step.
6. Saving the made changes – creation of a real ontology "Software Engineering" (part "Quality and complexity of the software. Metric analysis").

In [35], we have developed *method of activity of ontology-based intelligent agent for evaluating the initial stages of the software lifecycle*, which, on the basis of comparison of the obtained real ontology with the base ontology for non-functional characteristics of the software, forms the conclusion about the sufficiency or insufficiency of the information, and numerical assessments of the sufficiency level of the SRS information for defining the non-functional characteristics. Now we use this method for development of the IA for determining the sufficiency of metric information in the SRS. But the base ontology for such IA is the ontology "Software Engineering" (part "Quality and complexity of the software. Metric analysis"). In addition, the numerical assessment of the level of sufficiency of metric information in the SRS will be calculated by the formula (1):

$$D = \frac{k - \sum_{j=1}^{k} \frac{qmi_j}{qni_j}}{k}, \qquad (1)$$

where k – quantity of metrics ($k = 24$, because in [8] 24 metrics of quality and complexity, which are available at the early stages of lifecycle, were chosen), qmi_j – quantity of missing in the real SRS indicators for j-th metric, qni_j – quantity of necessary indicators for j-th metric (according to base ontology "Software Engineering" (part "Quality and complexity of the software. Metric analysis")).

Thus, the developed IS for determining the sufficiency of metric information in the SRS performs the parsing (semantic analysis) of the natural language SRS with the purpose of the search of the indicators, which are necessary for calculation of the software metrics, and also forms the conclusion about the sufficiency or insufficiency of metric information in the SRS, asesses the level of sufficiency of metric information and visually shows missing indicators with the distribution by the metrics, for which they are used.

4 Experiments with Intelligent System for Determining the Sufficiency of Metric Information in the Software Requirements Specifications

Intelligent system for determining the sufficiency of metric information in the SRS has implemented in the form of free software, which is available by the link – https://olp-project.herokuapp.com.

The user of the IS for determining the sufficiency of metric information in the SRS upload the SRS in pdf-format. IA for parsing the SRS on the search of metric information (indicators for calculation of metrics) performs parsing of the specification and generates the ontology for real software as a .owl file, which the user can download (Fig. 2).

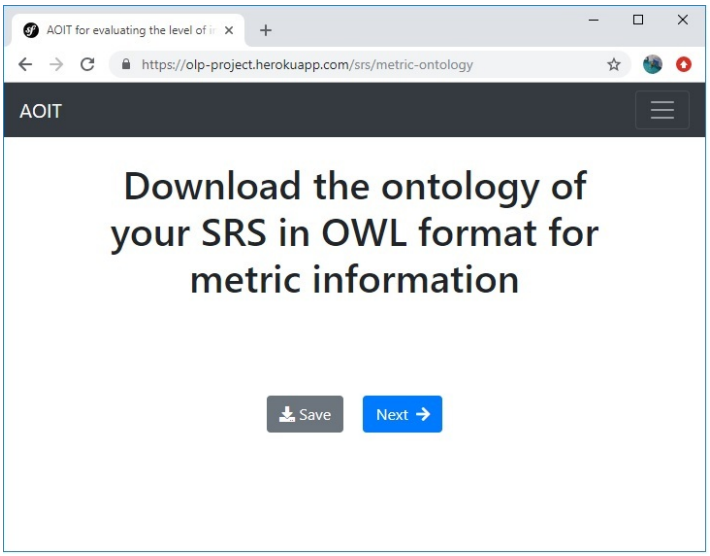

Fig. 2. Possibility of download of ontology for real SRS in OWL-format, which is provided by the intelligent system for determining the sufficiency of metric information in the SRS (results of IA for semantic parsing the SRS on the search of metric information).

Next, the IA for determining the sufficiency of metric information in the SRS performs a comparison of the ontology for real software with the base ontology and concludes about the sufficiency of metric information, namely: Count of missing indicators (without considering the number of uses of each indicator), Count of missing indicators (considering the number of uses of each indicator), Percent of missing indicators (without considering the number of uses of each indicator), Percent of missing indicators (considering the number of uses of each indicator), Total numerical evaluation of information sufficiency level for all metrics in SRS, and visualized list of missing indicators with distribution by the metrics.

For the experiments, two SRS of the information and analytical system for the accounting of therapeutic and diagnostic activities provided to the wounded during transportation were analyzed, which were developed by two different software companies at Khmelnitsky.

The level of sufficiency of metric information in the SRS1 was 49.72% when the 7 indicators without considering the number of uses of each indicator and 31 indicators considering the number of uses of each indicator were absent – Fig. 3.

The level of sufficiency of metric information in the SRS2 was 75.28% when the 14 indicators without considering the number of uses of each indicator and 20 indicators considering the number of uses of each indicator were absent – Fig. 4.

Obviously, the sufficiency of metric information in SRS2 is higher than the sufficiency of metric information in SRS1, and the number of missing indicators considering the number of uses of each indicator in SRS2 is lower than in SRS1 (while the number of missing indicators without considering the number of uses of each indicator in SRS2 is twice as high as in SRS1). These results are due to the fact that more important and priority are indicators that affect more than one metric.

On Fig. 5 the visualization of the missing indicators for determining the metrics of complexity with exact values at the design stage are represented with the distribution by metrics is represented.

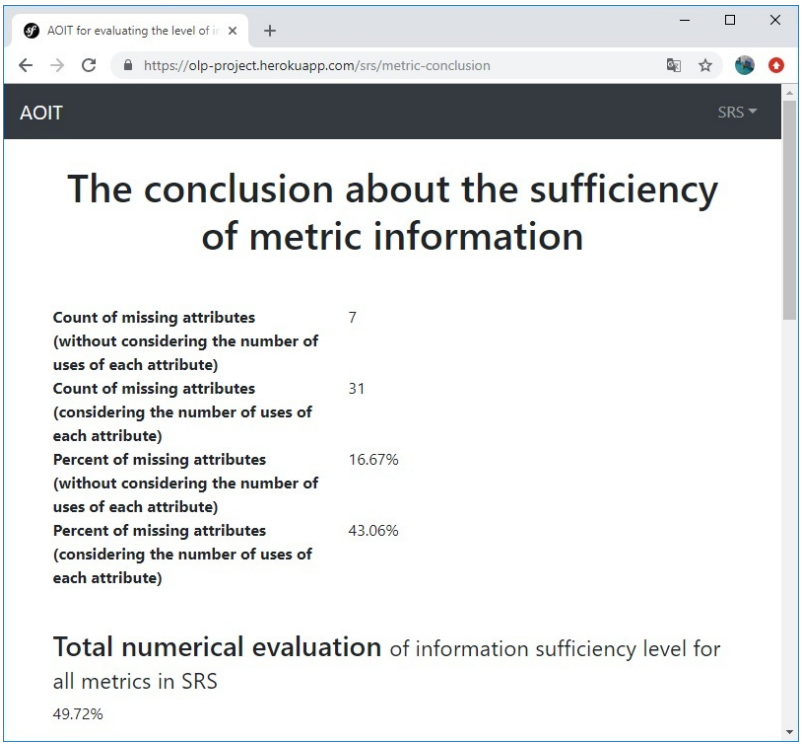

Fig. 3. Conclusion about the sufficiency of metric information in SRS1, which is provided by the intelligent system for determining the sufficiency of metric information in the SRS.

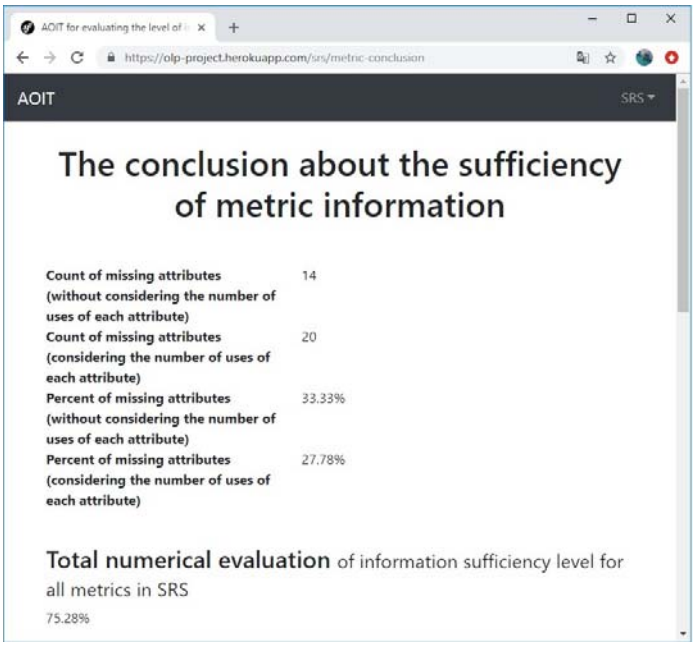

Fig. 4. Conclusion about the sufficiency of metric information in SRS2, which is provided by the intelligent system for determining the sufficiency of metric information in the SRS.

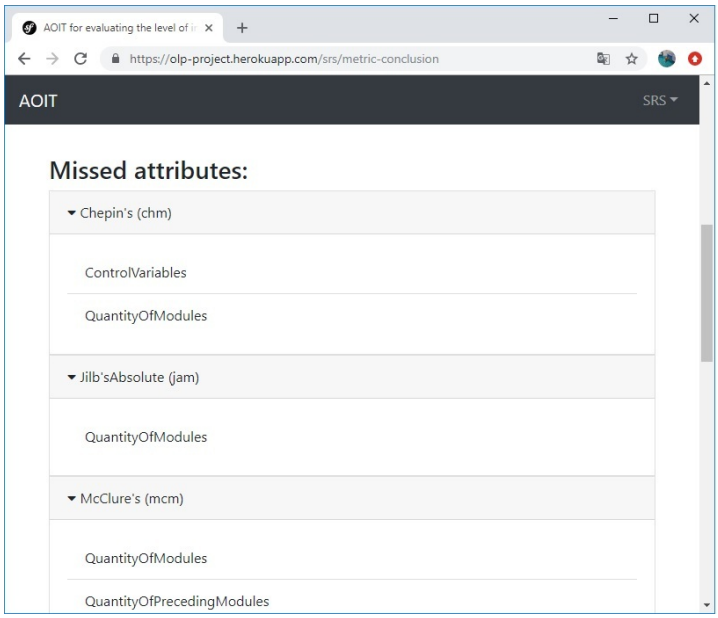

Fig. 5. Visualization of missing indicators (in SRS1) for determining the software complexity metrics with the exact values at the design stage, which is provided by the intelligent system for determining the sufficiency of metric information in the SRS.

The customer of the software of the information and analytical system for the accounting of therapeutic and diagnostic activities provided to the wounded during transportation considered the level of sufficiency equal 95% is acceptable (since medical software is the critical software). So the customer demanded the re-work of both SRS by their developers. After re-work, the SRS were again analyzed by the developed IS – Fig. 6, Fig. 7.

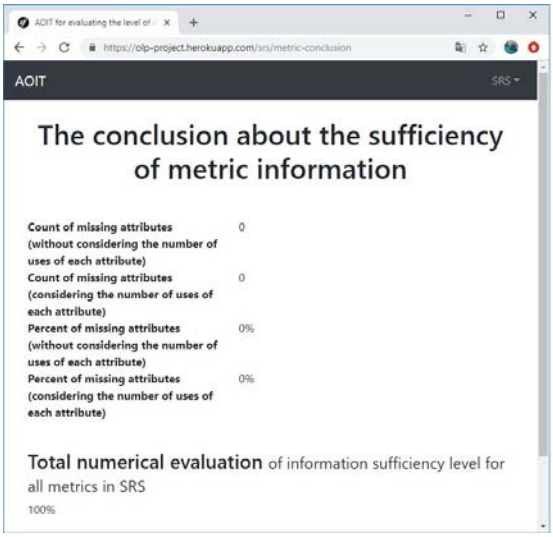

Fig. 6. Conclusion about the sufficiency of metric information in SRS1 (after re-work), which is provided by the intelligent system for determining the sufficiency of metric information in the SRS.

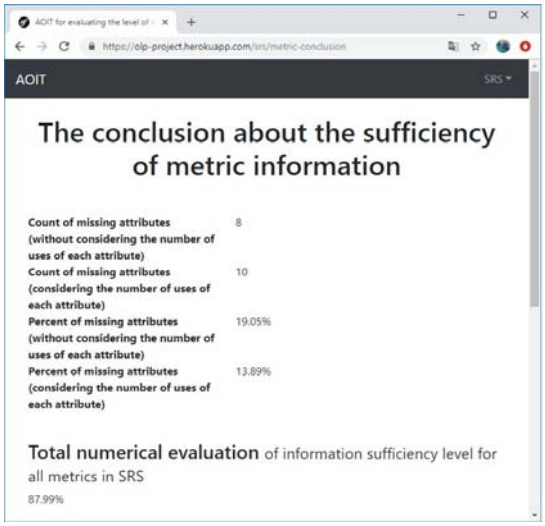

Fig. 7. Conclusion about the sufficiency of metric information in SRS2 (after re-work), which is provided by the intelligent system for determining the sufficiency of metric information in the SRS.

The level of sufficiency of the metric information in SRS1 after re-work is 100% (the SRS1 has all the necessary indicators for calculation of the metrics), and the level of sufficiency of the metric information in the SRS2 after re-work is 87.99% when 8 indicators without considering the number of uses of each indicator and 10 indicators considering the number of uses of each indicator are absent.

The customer of the software of the information and analytical system for the accounting of therapeutic and diagnostic activities provided to the wounded during transportation chose SRS1 with 100%-th level of sufficiency of metric information for further work on a software project.

The developed intelligent system for determining the sufficiency of metric information in the SRS has increased the sufficiency of information by 50.28% for SRS1 and by 12.71% for SRS2.

5 Conclusions

Nowadays the actual task is the calculation of the metrics' values for evaluating and predicting the quality and complexity of software at the early stages of the lifecycle. The conducted analysis of known methods and tools for performing metric analysis showed that most of these methods and tools are focused on the calculation of various metrics based on the analysis of the finished source code. Known methods and tools, which are aimed at early stages of lifecycle, are based on the analysis of the ready values of metrics and don't consider the possibility or impossibility of calculating such metrics on the basis of the available information in the SRS. Then, the actual task is assessing the sufficiency of metric information at the early stages of the software lifecycle, in particular, in the SRS.

In this paper the intelligent system for determining the sufficiency of metric information in the SRS is developed. This IS, based on the natural language processing of SRS, provides: the conclusion about the sufficiency of metric information in SRS, the numerical assessment of the sufficiency level of metric information in the SRS, visualization of missing indicators for metrics calculation.

The developed IS provides the increase in the sufficiency of information by 12.71-50.28% for the SRS of the software of the information and analytical system for the accounting of therapeutic and diagnostic activities provided to the wounded during transportation. Generally developed IS provides the increase in the sufficincy of metric information in the SRS to 100% – if it’s necessary (for critical software) or at the request of the customer.

The developed intelligent system for determining the sufficiency of metric information in the SRS can be used in the process of software development for government agencies, military formations and law enforcement agencies, for commercial organizations (for organizations-customers of the software – with the purpose of the assessment of the level of implementation of the initial stages of the lifecycle by the developers and with the purpose of the grounded choice of the SRS with the highest level of sufficiency of information). For organizations-developers,

the developed IS can also be used – with the purpose of automation of the process of verifying the sufficiency of information in the SRS.

The limitation of the developed IS is the possibility of assessment of the sufficiency of only metric information in the SRS – as the sufficiency of the indicators in the SRS for determining the 24 selected software metrics, which are available at the design stage. The further efforts of the authors will be directed for elimination of this limitation (in particular, for expanding the set of metrics and, respectively, indicators).

References

1. ISO/IEC 25010:2011. Systems and software engineering. Systems and software Quality Requirements and Evaluation (SQuaRE). System and software quality models (2011).
2. ISO/IEC 25030:2007. Software engineering. Software product Quality Requirements and Evaluation (SQuaRE). Quality requirements (2007).
3. ISO/IEC TR 19759:2015. Software Engineering. Guide to the software engineering body of knowledge (SWEBOK) (2015).
4. ISO 9000:2015. Quality management systems. Fundamentals and vocabulary (2015).
5. ISO 9001:2015. Quality management systems. Requirements (2015).
6. Maedche, A., Botzenhardt, A., Neer, L.: Software for people: fundamentals, trends and best practices (Management for professionals). Springer-Verlag Berlin Heidelberg, Berlin (2012).
7. ISO/IEC/IEEE 24765:2010. Systems and software engineering. Vocabulary (2010).
8. Pomorova, O., Hovorushchenko, T.: Research of Artificial Neural Network's Component of Software Quality Evaluation and Prediction Method. In: The 2011 IEEE 6-th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications Proceedings. Prague (2011).
9. Pomorova, O., Hovorushchenko, T.: The Way to Detection of Software Emergent Properties. In: The 2015 IEEE 8-th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications Proceedings. Warsaw (2015).
10. Pinciroli, F.: Improving Software Applications Quality by Considering the Contribution Relationship Among Quality Attributes. Procedia Computer Science. 83, 970-975 (2016).
11. Graphical Development Process Assistant: ProcePT, http://www.scope.gmd.de/projects/ProcePT/, last accessed 2019/03/19.
12. Cleanroom Software Engineering, https://www.tutorialride.com/software-engineering/cleanroom-software-engineering.htm, last accessed 2019/03/19.
13. Automatic Code Analysis with Logiscope Products, https://www.kalimetrix.com/logiscope, last accessed 2019/03/19.
14. PurifyPLUS > Dynamic Software Analysis, https://www.almtoolbox.com/purify.php, last accessed 2019/03/19.
15. Morse, J.: Expressive and efficient bounded modelchecking of concurrent software. University of Southampton, Southampton (2015).
16. DMS® Software Reengineering Toolkit™, https://www.semanticdesigns.com/Products/DMS/DMSToolkit.html, last accessed 2019/03/19.

17. Application Intelligence Platform, https://www.castsoftware.com/products/application-intelligence-platform, last accessed 2019/03/19.
18. ConQAT end of life, https://www.cqse.eu/en/blog/conqat-end-of-life/, last accessed 2019/03/19.
19. GrammaTech CodeSonar: Delivering resiliency for today's IoT devices, https://www.grammatech.com/products/codesonar, last accessed 2019/03/19.
20. Moose is a platform for software and data analysis, http://www.moosetechnology.org/, last accessed 2019/03/19.
21. Automated Software Testing Tools for Creating High Quality Software, https://www.parasoft.com/, last accessed 2019/03/19.
22. SideCI: Automated Code Review for GitHub, https://alternativeto.net/software/sideci/, last accessed 2019/03/19.
23. Sonargraph Product Family, https://www.hello2morrow.com/products/sonargraph, last accessed 2019/03/19.
24. SonarQube: The leading product for continuous code quality, https://www.sonarqube.org/, last accessed 2019/03/19.
25. CppCheck: A tool for static C/C++ code analysis, http://cppcheck.sourceforge.net/, last accessed 2019/03/19.
26. Zakariya, S., Belal, M.: Software quality management measured based code assessments. International Journal of Computer Science Trends and Technology. 3, 4, 263-268 (2015).
27. Shouman, M., Eldrandaly Kh., Tantawy, A.: Software quality assurance models and expert systems. In: The 9-th International Conference on Production Engineering, Design and Control Proceedings. Alexandria (2009).
28. Agüero, M., Madou, F., Esperón, G., López De Luise, D.: Artificial intelligence for software quality improvement. International Journal of Computer and Information Engineering. 4, 3, 399-404 (2010).
29. Farooq, S. U., Quadri, S., Ahmad, N.: Software measurements and metrics: role in effective software testing. International Journal of Engineering Science and Technology. 3, 1, 671-680 (2011).
30. Gururaj, H. L., Ramesh, B. BornBaby model for software synthesis: A program that can write programs. Data Analytics and Learning: Lecture Notes in Networks and Systems. 43, 403-412 (2019).
31. Ravi Kumar, T., Srinivasa Rao, T.: Software Defects Prediction based on ANN and Fuzzy logic using Software Metrics. International Journal of Applied Engineering Research. 12, 19, 8509-8517 (2017).
32. Azfal, W.: Search-based prediction of software quality: evaluations and comparisons. Blekinge Institute of Technology, Blekinge (2011).
33. Cruickshank, K. J.: A validation metrics framework for safety-critical software-intensive systems. Naval Postgraduate School, Monterey (2009).
34. Michael, J. B., Shing, M.-T., Cruickshank, K. J., Redmond, P. J.: Hazard Analysis and Validation Metrics Framework for System of Systems Software Safety. IEEE Systems Journal. 4, 2, 186–197 (2010).
35. Hovorushchenko, T., Pavlova, O.: Method of Activity of Ontology-Based Intelligent Agent for Evaluating the Initial Stages of the Software Lifecycle. Advances in Intelligent Systems and Computing. 836, 169-178 (2019).