CEUR-WS.org/Vol-2623/paper9.pdf

Approach to the Analysis of Software Requirements Specification on Its Structure Correctness

Artem Boyarchuk [1][0000-0001-7349-1371], Olga Pavlova [2][0000-0003-2905-0215], Mykyta Bodnar [3][0000-0002-8614-0682] and Ivan Lopatto [4][0000-0001-6886-2238]

[1] National Aerospace University "Kharkiv Aviation Institute", Kharkiv, Ukraine
[2, 3, 4] Khmelnytskyi National University, Khmelnytskyi, Ukraine
[1] a.boyarchuk@csn.khai.edu
[2] olya1607pavlova@gmail.com
[3] nikas.bodnar@gmail.com
[4] danloff@ukr.net

Abstract. During forming and formulating the requirements, it is important to comply with the standards that govern the software development process. The main basic standard for specifying the software requirements is ISO/IEC/IEEE 29148:2018, which regulates the structure and required items of the specification. Result of such analysis is: the known models, methods and tools don't solve the problem of software requirements specification analysis on its structure correctness (according to ISO/IEC/IEEE 29148:2018). In this paper, the authors have proposed the formalization of the structure of software requirements specification (according to ISO/IEC/IEEE 29148:2018) in the form of set-theoretical models of sections of the specification. The approach to the analysis of software requirements specification on its structure correctness (according to ISO/IEC/IEEE 29148:2018) was developed. This approach made it possible to perform a quick automated check of the software requirements specification on consideration of the above-defined specification's items. Such check allows making the automatic conclusion about the correctness/incorrectness of the structure of the specification, about the possibility of further work on the project according to such specification or about the need for re-work of the specification.

Keywords: Software, Software Requirements Specification, ISO/IEC/IEEE 29148:2018, Structure of Specification, Items of Specification, Correctness/Incorrectness of Specification's Structure.

1 Introduction

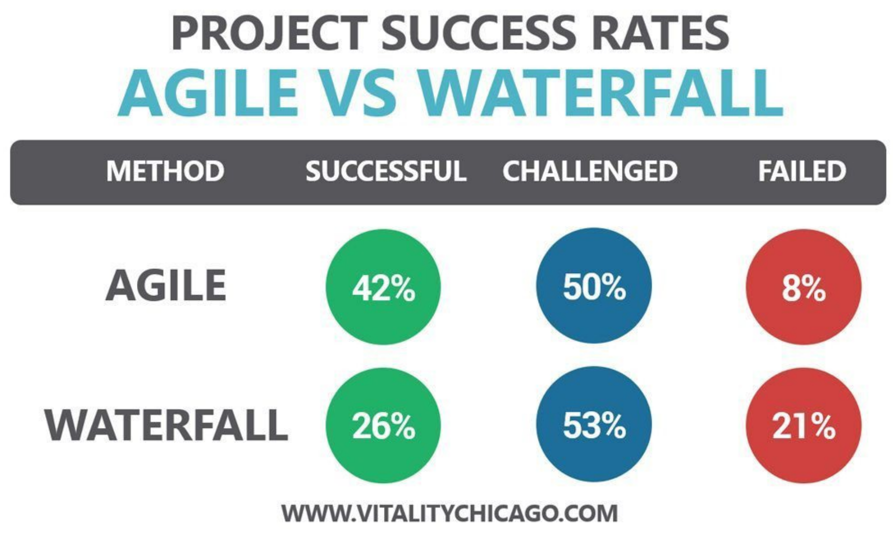
Nowadays, software systems are large and complex. They operate in complex and changing environments. The entire infrastructure (languages, libraries, operating systems, and computers) changes continuously. Often software systems are crucial for the operation of the entire business; hence there are quality, schedule and budget pressures. Software systems are part of larger systems (involving humans), which often

Copyright © 2020 for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0). IntelITSIS-2020

need to change along changing/developing a software system. Frequent failures of software projects today reassure us that the problems of software construction have not been solved. The main reason of software's failures and crashes is bad documentation at the design stage. Bad documentation leads to many bugs and decreases efficiency at every stage of a software’s development, causes the software project's failure or challenges [1].

In the past, the Standish Group International defined the success of software project considering the triple constraint (standard for the Project Management Institute) and classified software projects as successful, challenged or failed projects. The software project was a successful project if it met all three constraints (schedule, cost, and scope). The software project was a challenged project if it met two from three constraints (for example, on time and on the budget but with incomplete functionality). The software project was a failed project if it is cancelled before it is completed. Now the Standish Group International considers the customer value's delivery, compliance with the strategic objectives and satisfaction of the customer in determining the success of projects [2].

To date, software projects success rates are as follows – Figure 1 [2].



PROJECT SUCCESS RATES
AGILE VS WATERFALL

METHOD	SUCCESSFUL	CHALLENGED	FAILED
AGILE	42%	50%	8%
WATERFALL	26%	53%	21%

WWW.VITALITYCHICAGO.COM

Fig. 1. Software projects success rates in 2019 [2]

As Figure 1 shows, Agile-projects are more successful than Waterfall-projects, but Agile-methodology is not suitable for all types of software projects (for example, only Waterfall-methodology is acceptable for critical application software). In addition, among both Agile-projects and Waterfall-projects, half (1/2) of all projects are challenged, i.e. they usually have development time delays, budget overruns, or lack the required features.

Software project success factors are represented on Figure 2 [3]. Obviously, the clear requirements make up 13% in project success factors.

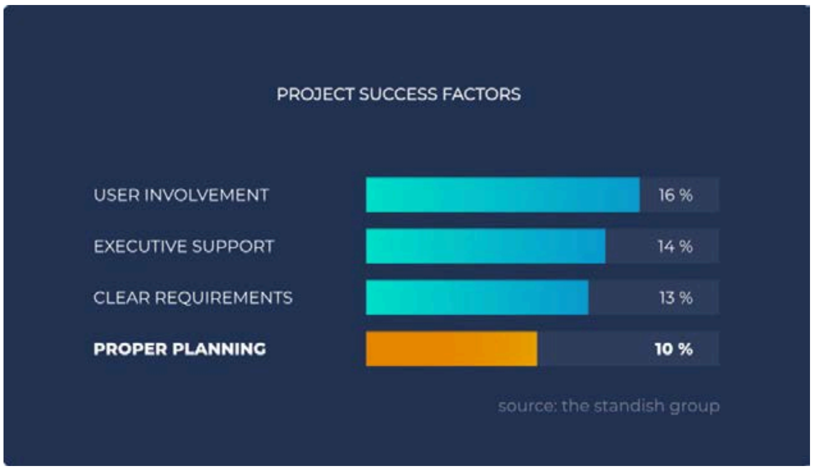


Fig. 2. Software project success factors for 2020 [3]

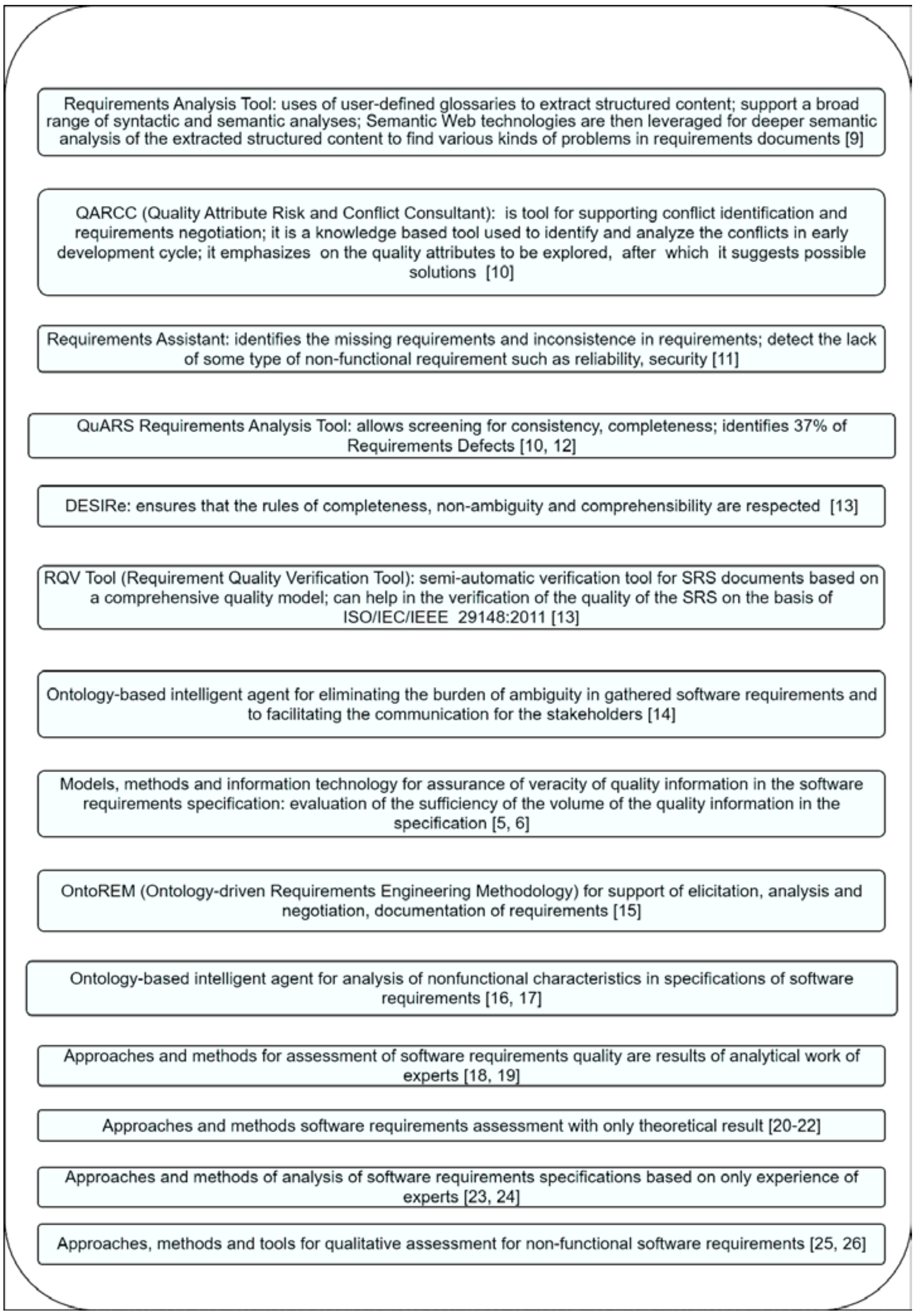
Many of the developers of software think that “documentation (in particular, requirements)” is a set of verbose, poorly structured, introductory descriptions which count the thousands of pages. Many of the traditional engineers consider the documentation as the primary design means. Developers of software consider the documentation as an afterthought because they don't know how software documents must be precisely composed. In fact, it should represent forethought, not an afterthought. Advantages of good documentation are facilitating the reuse of previous designs, improving the communication on requirements, increasing the efficiency of design reviews, facilitating the integration of individual modules, increasing the efficiency of inspection of code, increasing the efficiency of testing, and increasing the efficiency of corrections and improvements [1].

In [4] the authors proved that many software-related incidents and catastrophes due to erroneous requirements, due to their lack of clarity, due to incorrect specification structure, etc., therefore, the dependence of success of the software project implementation from the specification of requirements exists, so an in-depth analysis of the specification of software requirements is necessary. In [5] - [7] concept of estimating the information sufficiency in the specifications of the requirements for the software was developed, which has some actuality and necessity. The structure of the specifications (by standard ISO 29148 [8]) seriously affects the information sufficiency in the specifications of the requirements for the software.

Then analysis of software requirements specification on its structure correctness (according to ISO/IEC/IEEE 29148:2018) nowadays is *the actual task.*

2 Review of the Literature

Let's review the results of research to search the known tools, techniques and models, which are devoted for analysis of specifications of requirements for software – Figure 3.



Requirements Analysis Tool: uses of user-defined glossaries to extract structured content; support a broad range of syntactic and semantic analyses; Semantic Web technologies are then leveraged for deeper semantic analysis of the extracted structured content to find various kinds of problems in requirements documents [9]

QARCC (Quality Attribute Risk and Conflict Consultant): is tool for supporting conflict identification and requirements negotiation; it is a knowledge based tool used to identify and analyze the conflicts in early development cycle; it emphasizes on the quality attributes to be explored, after which it suggests possible solutions [10]

Requirements Assistant: identifies the missing requirements and inconsistence in requirements; detect the lack of some type of non-functional requirement such as reliability, security [11]

QuARS Requirements Analysis Tool: allows screening for consistency, completeness; identifies 37% of Requirements Defects [10, 12]

DESIRe: ensures that the rules of completeness, non-ambiguity and comprehensibility are respected [13]

RQV Tool (Requirement Quality Verification Tool): semi-automatic verification tool for SRS documents based on a comprehensive quality model; can help in the verification of the quality of the SRS on the basis of ISO/IEC/IEEE 29148:2011 [13]

Ontology-based intelligent agent for eliminating the burden of ambiguity in gathered software requirements and to facilitating the communication for the stakeholders [14]

Models, methods and information technology for assurance of veracity of quality information in the software requirements specification: evaluation of the sufficiency of the volume of the quality information in the specification [5, 6]

OntoREM (Ontology-driven Requirements Engineering Methodology) for support of elicitation, analysis and negotiation, documentation of requirements [15]

Ontology-based intelligent agent for analysis of nonfunctional characteristics in specifications of software requirements [16, 17]

Approaches and methods for assessment of software requirements quality are results of analytical work of experts [18, 19]

Approaches and methods software requirements assessment with only theoretical result [20-22]

Approaches and methods of analysis of software requirements specifications based on only experience of experts [23, 24]

Approaches, methods and tools for qualitative assessment for non-functional software requirements [25, 26]

Fig. 3. Tools, techniques and models for analysis of specifications of requirements for software

Therefore, the results of such analysis - the known tools, techniques and models don't solve the problem of structure correctness (by ISO/IEC/IEEE 29148:2018) of specifications of requirements for the software. Only the RQV Tool (Requirement Quality Verification Tool) [13] can help in the verification of the quality of the SRS on the basis of ISO/IEC/IEEE 29148:2011. In addition, all tools, techniques and models represent the different approaches and are not integrated into a single whole, i. e. today there is no single approach to the analysis of software requirements specification on its structure correctness (according to ISO/IEC/IEEE 29148:2018).

Considering the defined urgency and importance of the problem of determination of structure correctness of specification of requirements for the software, *the goal of such study* is the development of the approach to the determination of structure correctness (by ISO/IEC/IEEE 29148:2018) of specification of requirements for the software.

3 Approach to the Analysis of Software Requirements Specification on Its Structure Correctness (according to ISO/IEC/IEEE 29148:2018)

Given the structure of the specification of the software requirements in accordance with ISO 29148:2018, let's represent the specification in the following formalized form:

$$SRS=<S\_1,\ldots,S\_19>, \quad (1)$$

where S_1 – section "Purpose" of the software requirements specification, S_2 – section "Scope", S_3 – section "Product perspective", S_4 – section "Product functions", S_5 – section "User chararcteristics", S_6 – section "Limitations", S_7 – section "Assumptions and dependencies", S_8 – section "Apportioning of requirements", S_9 – section "Specific requirements", S_10 – section "External interfaces", S_11 – section "Functions", S_12 – section "Usability requirements", S_13 – section "Performance requirements", S_14 – section "Logical database requirements", S_15 – section "Design constraints", S_16 – section "Standards compliance", S_17 – section "Software system attributes", S_18 – section "Verification", S_19 – section "Supporting information" of the software requirements specification.

Sections of specification of the requirements for the software consist of a set of items (by ISO 29148) and have the following set-theoretical form:

$$S\_1 = \{ps\}, \quad (2)$$

$$S\_2 = \{isp, spd, spb, spo, spg, cssh\}, \quad (3)$$

$$S\_3 = \{srrp, sosi, soui, sohi, soswi, soci, som, soo, sosar\}, \quad (4)$$

$$S\_4 = \{mf\}, \quad (5)$$

$$S\_5 = \{gcigu, gciu\}, \tag{6}$$

$$S\_6 = \{rp, hwl, ioa, plo, af, cf, holr, shp, quar, ca, ssc, pmc\}, \tag{7}$$

$$S\_7 = \{far\}, \tag{8}$$

$$S\_8 = \{asrse, crtf, crte, rduf\}, \tag{9}$$

$$S\_9 = \{rlds, iss, oss, fss\}, \tag{10}$$

$$S\_10 = \{ni, dp, sido, vrat, um, tg, roio, sfo, wfo, df, cmf, emsg\}, \tag{11}$$

$$S\_11 = \{faapi, fapgo, vci, eso, ras, ep, rsoi\}, \tag{12}$$

$$S\_12 = \{ubr, mec, mefc, msc\}, \tag{13}$$

$$S\_13 = \{snr, dnr, nts, nssu, athi, nota, not, adnw, adpw, prq\}, \tag{14}$$

$$S\_14 = \{tiuvf, fqu, asc, der, ics, drr\}, \tag{15}$$

$$S\_15 = \{ces, crr, cpl\}, \tag{16}$$

$$S\_16 = \{rf, dn, ap, at\}, \tag{17}$$

$$\begin{gathered} S\_17 = \{rb, avb, scr, cct, slhd, fdm, csa, dicv, dp, \\ mb, pb, pehdc, pchd, uppl, upcls, upos\}, \end{gathered} \tag{18}$$

$$S\_18 = \{va, mqs\}, \tag{19}$$

$$S\_19 = \{siof, dcas, rus, sbi, dpss, spi\}, \tag{20}$$

where ps – purpose of software; isp – the software product's identifying, spd – what software product will provide, spb – benefit of the software product, spo – objectives of the software product, spg – goals of the software product, cssh – consistency with the same statements in high-level specifications; srrp – software system's relationship to other related products, sosi – software operation within the system interfaces, soui – software operation within the user interfaces, sohi – software operation within the hardware interfaces, soswi – software operation within the software interfaces, soci – software operation within the communications interfacse, som – software operation within the memory, soo – software operation within the operations, sosar – software operation within the site adaprion requirements; mf – major future functions of the software; gcigu – general characteristics of the software product's groups of users, gciu – general characteristics which influence usability; rp – regulatory policies, hwl – limitations of hardware, ioa – interfaces to other applications, plo – parallel operation, af – audit functions, cf – control functions, holr – requirements of higher-order language, shp – protocols of signal handshake, quar – quality requirements, ca – criticality of the application, ssc – considerations of safety and security, pmc – physical/mental considerations; far – factors which influence the requirements; asrse – distribution the requirements to elements of software, crtf – cross-reference table by function, crte – cross-reference table by element, rduf – requirements that can be transferred in software's future versions; rlds – requirements to a detail sufficient level, iss – inputs into the software system, oss – outputs from the software system, fss – functions of the software system in response to the input or in support of the output; ni – item's name, dp – purpose's description, sido – input's source or output's destination, vrat – valid range, accuracy, and/or tolerance, um – measuring units, tg – timing, roio – relationships to other inputs/outputs, sfo – formats/organization of screen, wfo – formats/organization of window, df – formats of data, cmf – formats of command, emsg – endmessages; faapi – basic actions that must take place in the software when receiving and processing inputs, fapgo – basic actions that must occur in the software when processing and generating outputs, vci – checks of validity on the inputs, eso – exact sequence of operations, ras – responses to abnormal situations, ep – parameters' effect, rsoi – relationship of inputs and outputs; ubr – usability requirements, mec – measurable criteria of effectiveness in specific use contexts, mefc – measurable criteria of efficiency in specific use contexts, msc – measurable criteria of satisfaction in specific use contexts; snr – static numerical requirements, dnr – dynamic numerical requirements, nts – number of supported terminals, nssu – number of supported simultaneous users, athi – amount and type of handled information, nota – numbers of transactions, not – number of tasks, adnw – amount of the processed data within certain time periods for normal workload conditions, adpw – amount of the processed data within certain time periods for peak workload conditions, prq – requirements of performance; tiuvf – types of used information, fqu – frequency of use, asc – accessing the capabilities, der – entities of data and their relationships, ics – constraints of integrity, drr – requirements of data retention; ces – the software system design's constraints which are imposed by external standards, crr – the software system design's constraints which are imposed by requlatory requirements, cpl – the software system design's constraints which are imposed by project limitations; rf – report format, dn – data naming, ap – accounting procedures, at – audit tracing; rb – reliability, avb – availability, scr – security, cct – certain techniques of cryptographic, slhd – data sets of specific log or history, fdm – certain functions to different modules, csa – communications between some areas of the software product, dicv – integrity of data for critical variables, dp – privacy of data, mb – maintainability, pb – portability, pehdc – the percentage of elements with host-dependent code, pchd – the percentage of host-dependent code, uppl – portable language use, upcls – particular compiler or language subset use, upos – operating system use; va – approaches for verification, mqs – methods for qualifying the software; siof – sample input/output formats, dcas – cost analysis studies' descriptions, rus – user surveys' results, sbi – supporting or background information that can help the readers of the SRS, dpss – description of the problems which will be solved by the software, spi – special packaging instructions for the code and the media for security, export, initial loading.

Approach to the analysis of software requirements specification on its structure correctness (according to ISO/IEC/IEEE 29148:2018) is represented on Figure 4 (on the basis of concept, which is proposed in [5]-[7]).

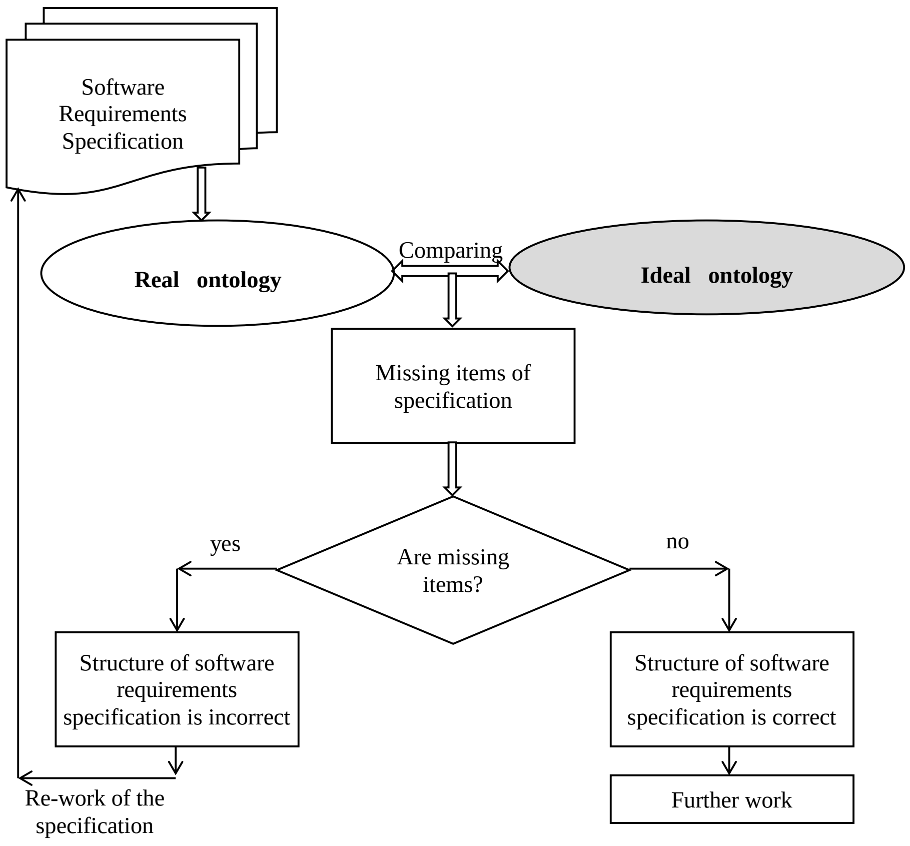

Software Requirements Specification
Real ontology
Comparing
Ideal ontology
Missing items of specification
Are missing items?
yes
no
Structure of software requirements specification is incorrect
Structure of software requirements specification is correct
Re-work of the specification
Further work

Fig. 4. Approach to the analysis of software requirements specification on its structure correctness (according to ISO/IEC/IEEE 29148:2018)

The equations (2) - (20) represent all the necessary items of the software requirements specification in terms of ISO/IEC/IEEE 29148:2018, structured by the sections of the specification. Therefore, for the structure of the specification of requirements to be recognized as correct, it is necessary that the specification contains all the listed items. For acceleration and automation of such a procedure of check for all items availability, it is proposed to develop an ideal ontology based on the equations (2) - (20), as well as to develop a real ontology based on each analyzed specification. Next, a comparison of the two ontologies (ideal and real) will result in missing items of specification being set. If such missing items are available, then the structure of specification is incorrect and re-work of the specification is proposed. If there are no such missing items, then the structure of specification is correct and further work is proposed.

The use of ontologies in this approach to the analysis of software requirements specification on its structure correctness (according to ISO/IEC/IEEE 29148:2018) provides the automation of such analysis.

4 Experiment, Results and Discussions

Checking the correctness of the structure of the finished (in mind of the developers) specification of requirements for a software system for providing the resilience of computer systems was conducted with the help of the developed approach. This checking identified that the prepared document has not the following items: “software operation within the system interfaces”, “factors which influence the requirements”, “cross-reference table by function”, “dynamic numerical requirements”, “requirements that can be transferred in software's future versions”, “basic actions that must take place in the software when receiving and processing inputs”, “accessing the capabilities”, “percentage of elements with host-dependent code”, “types of used information”, “requirements of performance”, “communications between some areas of the software product”, “portable language use”, “particular compiler or language subset use”, “sample input/output formats”, “supporting or background information that can help the readers of the SRS”. It was concluded that "Structure of software requirements specification is incorrect" and re-work of the specification is proposed. The developers finalized the specification, re-analyzed the specification using the developed approach, resulting in the conclusion that there are no missing items, and thus concluded that "Structure of software requirements specification is correct" and further work is proposed.

The proposed set-theoretical models of software requirements specification's sections (according to ISO/IEC/IEEE 29148:2018), as well as the approach to the determination of structure correctness (by ISO/IEC/IEEE 29148:2018) of specification of requirements for the software, have provided the ability of quick and automated checking the correctness of the structure of software requirements specification, considering the availability of specification's items. Such checking gives the automated decision about the correctness/incorrectness of the structure of specification, about the possibility/impossibility of continuation of work on the project according to such specification, about the need/uselessness of re-work of the specification.

The conducted experiment showed the effectiveness of the proposed approach to the determination of structure correctness (by ISO/IEC/IEEE 29148:2018) of specification of requirements for the software.

5 Conclusions

During developing the software, software organizations must be guided by standards for both software development and evaluation processes. During forming and formulating the requirements, it is important to comply with the standards that govern the software development process. The main basic standard for specifying the software requirements is ISO/IEC/IEEE 29148:2018, which regulates the structure and required items of the specification. The conducted analysis has shown that the known models, methods and tools for analysis of software requirements specification do not solve the problem of analysis of software requirements specification on its structure correctness (according to ISO/IEC/IEEE 29148:2018).

Given the regulated structure and mandatory items of specification, in this paper, the authors have proposed the formalization of the structure of software requirements specification (according to ISO/IEC/IEEE 29148:2018) in the form of set-theoretical models of sections of the specification.

Based on the proposed formalization of the specification, the approach to the analysis of software requirements specification on its structure correctness (according to ISO/IEC/IEEE 29148:2018) was developed. This approach made it possible to perform a quick automated check of the software requirements specification on consideration of the above-defined specification's items. Such checking makes the automatic conclusion about the correctness/incorrectness of the specification's structure, about the possibility of continuation of the work on the project according to such specification, about the need/uselessness of re-work of the specification.

The prospective research of authors will be devoted to the development of the ideal ontology on the basis of the developed set-theoretical models of specification's sections; to the development of a method of activity and to the realization of the ontology-based intelligent agent, which will work on the basis of the developed approach and will perform automatic verification of the correctness of structure of the specification (by ISO/IEC/IEEE 29148:2018).

References

1. Leonard, J.: CSCE 606: Introduction, 2019, https://slideplayer.com/slide/15545897/, last accessed 2020/04/10.
2. Mersino, A.: Agile Project Success Rates are 2X Higher than Traditional Projects (2019), 2018, https://vitalitychicago.com/blog/agile-projects-are-more-successful-traditional-projects/, last accessed 2020/04/10.
3. 4-Step Project Plan for 2020, 2019, http://projexs.io/project-plan-made-easy/, last accessed 2020/04/10.
4. 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).
5. Hovorushchenko, T.: Methodology of Evaluating the Sufficiency of Information for Software Quality Assessment According to ISO 25010. Journal of Information and Organizational Sciences 42(1), 63-85 (2018).
6. Hovorushchenko, T.: Information Technology for Assurance of Veracity of Quality Information in the Software Requirements Specification. Advances in Intelligent Systems and Computing 689, 166-185 (2018).
7. Hovorushchenko, T., Pomorova, O.: Ontological Approach to the Assessment of Information Sufficiency for Software Quality Determination. CEUR-WS 1614, 332-348 (2016).
8. ISO/IEC/IEEE 29148:2018. Systems and software engineering. Life cycle processes. Requirements engineering (2018).
9. Verma, K., Kass, A.: Requirements Analysis Tool: A Tool for Automatically Analyzing Software Requirements Documents. Lecture Notes in Computer Science 5318, 751-763 (2008).
10. Mahmood, H., Rehman, M. S.: Tools for software engineers. International Journal of Research in Engineering & Technology 3 (10), 75-86 (2015).
11. Jones., V., Murray, J.: Evaluation of current requirements analysis tools capabilities for IV&V in the requirements analysis phase, https://www.slideserve.com/shlomo/evaluation-

of-current-requirements-analysis-tools-capabilities-for-ivv-in-the-requirements-analysis-phase, last accessed 2020/04/10.

12. Raffo, D. M., Ferguson, R.: Evaluating the Impact of The QuARS Requirements Analysis Tool Using Simulation, https://pdfs.semanticscholar.org/7e7d/4e6f5ab13d00ca57c87711e30cd080730f34.pdf, last accessed 2020/04/10.
13. Konig, F., Ballejos, L., Ale, M.: A semi-automatic verification tool for software requirements specification documents. In: Simposio Argentino de Ingeniería de Software Proceedings. Sociedad Argentina de Informática e Investigación Operativa (2017).
14. Ossowska, K., Szewc, L., Weichbroth, P., Garnik, I., Sikorski, M.: Exploring an ontological approach for user requirements elicitation in the design of online virtual agents. Information Systems: Development, Research, Applications, Education 264, 40-55 (2017).
15. Meziane, F., Vadera, S.: Artificial Intelligence Applications for Improved Software Engineering Development: New Prospects. Advances in Intelligent Information Technologies 278-299 (2010).
16. 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).
17. Hovorushchenko, T., Pavlova, O., Bodnar, M.: Development of an Intelligent Agent for Analysis of Nonfunctional Characteristics in Specifications of Software Requirements. Eastern-European Journal of Enterprise Technologies 1(2), 6-17 (2019).
18. Ahmad, S., Asmai, S.A.: Measuring software requirements quality following negotiation through empirical study. International Journal of Applied Engineering Research 11(6), 4190-4196 (2016).
19. Thitisathienkul, P., Prompoon, N.: Quality Assessment Method for Software Requirements Specifications Based on Document Characteristics and Its Structure. In: The Second International Conference on Trustworthy Systems and their Applications Proceedings. Hualien (2015).
20. Jain, H.K., Vitharana, P., Zahedi, F.: An assessment model for requirements identification in componentbased software development. Association for Computing Machinery New York NY United States 34(4), 48-63 (2003).
21. Audytra, H., Hendradjaya, B., Sunindyo, W.D.: A proposal for quality assessment model for software requirements specification in Indonesian language for e-Government. In International Conference on Data and Software Engineering Proceedings. Denpasar (2016).
22. Nazaruka, E., Osis, J.: The Formal Reference Model for Software Requirements. Communications in Computer and Information Science 1023, 352-372 (2018).
23. Tan, H., Ismail, M., Tarasov, V., Adlemo, A., Johansson, M.: Development and Evaluation of a Software Requirements Ontology. In: The 7th International Workshop on Software Knowledge Proceedings. Porto (2016).
24. Alshazly, A., Elfatatry, A., Abougabal, M.S.: Detecting defects in software requirements specification. Alexandria Engineering Journal 53(3), 513-527 (2014).
25. Kohl, M.A., Baum, K., Langer, M., Oster, D., Speith, T., Bohlender, D.: Explainability as a Non-Functional Requirement. In: IEEE 27th International Requirements Engineering Conference Proceedings. Jeju Island (2019).
26. Mahalakshmi, K., Prabhakar, R.: Performance Evaluation of Non Functional Requirements. Global Journal of Computer Science and Technology Software & Data Engineering 13(8), 15-19 (2013).