prof. Ing. Tetiana Hovorushchenko, DrSc.
Ing. Olga Pavlova
doc. Ing. Artem Boyarchuk, PhD.
doc. Ing. Miroslav Kvassay, PhD.
doc. Ing. Yelyzaveta Hnatchuk, PhD.
doc. Ing. Dmytro Medzatyi, PhD.

Intelligent Information-Analytical Technologies for Improving the Software Quality by Assessing the Sufficiency of Information at Initial Stages of the Life Cycle

Published by University of Žilina
EDIS-Žilina University Publisher
2020

Scientific editor:
doc. Ing. Marek Kvet, PhD.
Faculty of Management Science and Informatics
University of Žilina
Univerzitná 8215/1
010 26 Žilina
Slovakia

Reviewers:
prof. Ing. Dmytro Maevsky, DrSc.
Department of Electromechanical Engineering
Odessa National Polytechnic University
Shevchenko Avenue 1
65044 Odessa
Ukraine

prof. Ing. Sergiy Ostapov, DrSc.
Department of Computer Systems Software
Yuriy Fed'kovych Chernivtsi National University
Kotsubynskyi Street 2
58012 Chernivtsi
Ukraine

The authors are responsible for the professional, linguistic and technical level of the publication.

Published by University of Žilina/EDIS-Žilina University Publisher

ISBN 978-80-554-1729-5

Contents

Foreword

At the current stage of development and implementation of information technology in various areas of human activity, decisive changes are taking place, as there are powerful technical resources for the accumulation and processing of large amounts of information. However, the application of known methods and tools to process such arrays of information does not meet the expectations of developers, leads to overuse of resources, loss of significant information and conflicts between customer expectations and results.

At the same time, such areas of intellectualization of information and data processing as Machine Learning, Cognitive Computing, Big Data Processing, Deep Learning, Semantic WEB Concept, etc. are rapidly developed, which allow solving new classes of problems based on available information resources [1-14].

All of the above are prerequisites for the transition to a new qualitative level of information processing, and, accordingly, for the development of a methodology for creating and implementing new-generation information technologies.

However, the specifics and features of the subject areas for which information technology is developed, significantly affect the content and methods of information processing, so the expectation of universal approaches to creating effective information technology for different industries today is premature. An approach based on the study of the characteristics and features of subject areas and the development of new information technologies specifically for specific domains remains justified.

The field of software engineering needs special attention in the direction of development and implementation of effective information technologies, in particular, to solve the problem of software quality assurance. Developing high-quality software is a key factor in its effective use and one of the main needs of customers. The need for quality assurance is based on the fact that software errors and failures lead to disasters that lead to human casualties, environmental cataclysms, significant time losses and financial losses.

Statistics show that today there are problems in the field of software quality assurance [15-28] – large projects are still carried out behind schedule or in excess of cost estimates, developed software often does not have the necessary functionality, its performance is low, and the quality does not suit consumers. Today, the world spends more than $ 250 billion annually on the development of approximately 175 thousand software projects. The average cost of the project for a large company is 2.322 million USD, for a medium company – 1.313 million USD, and for a small company – 434 thousand USD [20, 21]. At the same time, a significant number of software projects are unsuccessful (with overuse of time, money, insufficient functionality or those that are cancelled until completion and are never used). On average, only 16-29% of program projects are implemented within the planned time and budget (for large companies – 9-16% of projects); projects implemented by the largest American companies have only about 42% of the required capabilities and functions [20, 21] – Fig. F.1 [26].

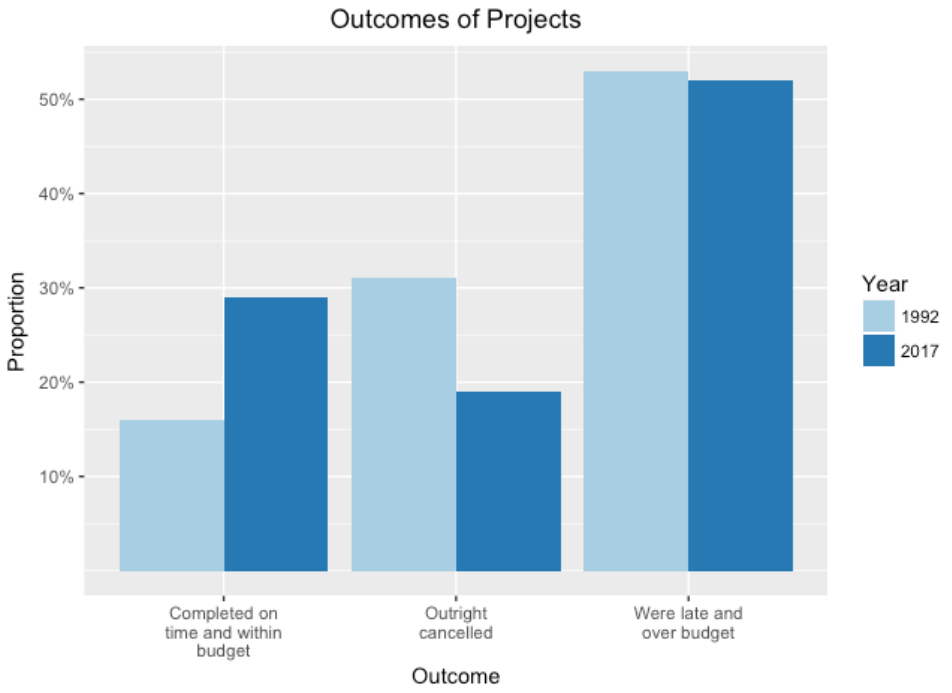

Fig. F.1. Comparative statistics on the success of software projects in 1992 and 2017, according to The Standish Group International [26]

A significant number of errors are made in the software at the stage of formation and formulation of requirements [15, 27] – according to statistics, 56% of all defects of software projects are introduced at the stage of formation and formulation of requirements; about 50% of defects in requirements are the result of poorly written, unclear, ambiguous or incorrect requirements; the other 50% are due to incomplete specifications (incomplete and omitted requirements) [29]. The vast majority of software-related accidents are caused by erroneous requirements rather than coding errors. The earlier the defect is detected (error, violation, defect, malfunction), the cheaper it will be to correct it. According to statistics, the cost of correcting defects and software errors made in the early stages of the life cycle increases exponentially with each subsequent stage of the project life cycle (Fig. F.2) [29]. The cost of correcting incorrect requirements in the specification, which are identified after the release of the product, is almost 100 times higher than the cost of correcting the shortcomings of the specification, which were identified at earlier stages, for example, at the stage of formation and formulation of requirements [30]. The risks of the insufficiently worked-out stage of formation and formulation of requirements are non-compliance with project deadlines and financial overspending, which can lead to the closure of the project, and even the collapse of the software company due to its financial instability.

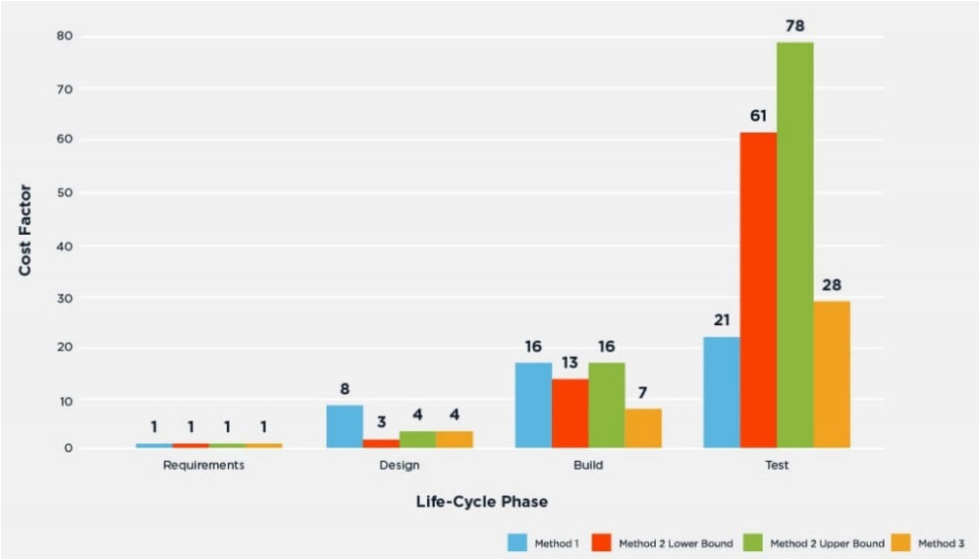

Fig. F.2. Comparison of system cost factors [29]

Thus, the critical impact on software projects and the success of their implementation are issues related to the analysis and evaluation of the initial stages of the life cycle. Today, when the number of high-budget software projects is growing rapidly, analysis of the specification of software requirements is *actual*. The possibility of automated assessment of the initial stages of the software life cycle, in particular, identifying and eliminating shortcomings of the initial stages of the software life cycle and the facts of the insufficiency of information, which is relevant to them, is actual too (moreover special attention needs to be paid to information about the non-functional characteristics of the software).

In the process of formulating requirements, information losses occur due to incomplete and different understanding of information needs and context – especially such losses are significant for software projects developed at the intersection of subject areas (e.g., software for medicine), when as standards for software development, and standards of the subject area should be considered. Such a number of standards is difficult to implement, and even more difficult to assess the extent to which the recommendations of these standards have been taken into account.

The software development life cycle begins with the formation of a set of requirements and software requirements specification (SRS) based on them. Software projects, the specification of the requirements of which contains insufficient, inaccurate, incomplete and contradictory information, cannot be successfully implemented [27].

Therefore, to ensure the quality of the software, it is necessary to study the requirements for the software, in order to identify and eliminate problems and shortcomings in the initial stages of the software life cycle and to identify the facts of insufficiency of

information related to them. In the course of such a study, it is necessary to assess how fully the software requirements specifications reflect information on the functions and limitations of future software. Particular attention needs to be paid to functions that characterize non-functional characteristics (for example, software quality characteristics).

Today, two key approaches to software quality assessment are: the SQuaRE model (ISO 25010:2011 standard) and based on the results of the metric analysis. Software quality assessment according to ISO 25010:2011 [31] is performed as follows: based on the quality attributes specified in ISO 25023:2016 [32], sub-characteristics and characteristics of quality are evaluated, which, in turn, provide a comprehensive assessment of software quality. Quality assessment on the basis of metric information processing is as follows: on the basis of indicators, the values of quality metrics are calculated, which, in turn, make it possible to calculate a comprehensive assessment of software quality.

Then *the quality information* in the software requirements specification consists of the attributes and quality indicators defined in the specification.

Sufficiency of information on quality in the specification of software requirements will be understood as the presence in the specification of all information elements (attributes and indicators), which is necessary to determine the quality.

Currently, the evaluation of attributes/indicators to determine the quality of the software is only at the stage of quality evaluation for the finished program code [33-35]. But all the necessary attributes/indicators of quality are laid down in the specification of software requirements, i.e. already on the basis of the specification of requirements, it is possible to assess the sufficiency of information for further determination and achievement of software quality [36-40].

If some attributes and/or metrics are missing, then the information in the requirements specification is insufficiency to determine the quality of the software. To eliminate the lack of information in the specification of software requirements, it is necessary to generate repeated requests for requirements to the developers of specifications. On the basis of such a request, the developers of the specifications must make the necessary additions to the requirements that regulate the quality characteristics.

It was found that quality information is convenient to present in the form of ontologies that allow to display causal relationships between concepts, integrate data, knowledge and requirements for software quality, as well as to identify missing attributes and/or indicators in the specification [41-47].

The analysis of known models, methods and tools showed that they do not solve the problem of assessing the sufficiency of quality information in the specifications of software requirements. All of them belong to different methodological approaches and are not integrated with each other, i.e. there is currently no methodology and information technology to improve the quality of software by assessing the sufficiency of information in the early stages of the life cycle.

The need to ensure the quality of software, the presence of information losses in the formation and formulation of software requirements, the need to identify and eliminate insufficiency of information at the initial stages of the software life cycle creates *an urgent*

scientific and applied problem, one way to solve which is to develop intelligent information technology for improving the quality of software by assessing the sufficiency of information in the early stages of the life cycle, which will provide the ability to identify lack of quality information in the specifications of the requirements in the early stages of the software life cycle and the ability to generate requests to supplement requirements by such information, which together will allow to assess the current situation and provide the appropriate level of quality at later stages of the software life cycle.

The monograph is devoted to solving the current scientific and applied problem of improving the quality of software in the early stages of the life cycle by assessing the sufficiency of information on quality in the specifications of software requirements. Developed intelligent information-analytical technologies for improving the quality of software by assessing the sufficiency of information in the early stages of the life cycle: will increase the level of sufficiency of information of requirements for determining the quality of software, thereby reducing the gap in knowledge about software projects; will provide a conclusion on the sufficiency of information on quality in the specification of requirements; will determine the priority of supplementing the specification with the necessary information (in case of insufficient information); will provide a quantitative assessment of the level of sufficiency of the quality information, which is available in the specification; will provide the opportunity to process quality information in the software requirements specifications by intelligent agents, without the participation of specialists, which will eliminate the subjective influence of specialists and the safety of this information in the software company in case of dismissal of specialists. The developed information-analytical technologies will automatically process the existing knowledge (quality requirements from the specification) and will form new knowledge (conclusions about the sufficiency of information, about the level of information sufficiency, recommendations for improving the sufficiency of information in the specification requirements). Developed intelligent information-analytical technologies will increase the sufficiency of information on quality in the specifications of software requirements to 100% – if it's necessary (for critical application systems) or at the request of the customer.

In general, the monograph is devoted to solving the current scientific and applied problem of improving the software quality in the early stages of the life cycle by assessing the sufficiency of information on quality in the specifications of software requirements. The developed intelligent information-analytical technologies automatically process the existing knowledge (quality requirements from the specification) and generate new knowledge (conclusions about the sufficiency of information, the level of information sufficiency, recommendations for improving the sufficiency of information in the specification requirements). The developed intelligent information-analytical technologies allow increasing the sufficiency of information on quality in the specifications of software requirements up to 100% – if it's necessary (for systems of critical application) or at the request of the customer.

The researches presented in the monograph were carried out within the framework of the state budget research of Khmelnytskyi National University "Agent-oriented system for

improving the security and quality of computer systems' software" (state registration number 0119U100662) and project of the Slovak Research and Development Agency “New methods development for reliability analysis of complex system” (reg. no. APVV-18-0027).

The monograph is intended for researchers and engineers involved in software design and development, as well as for lecturers, PhD students and students of higher education institutions.

The authors will be grateful for the feedback on the monograph and wishes for the development of further research.

Team of authors

Chapter 1. State of the Art

1.1. Analysis of the Impact of Information in the Specification of Requirements on Software Quality

Almost all areas of human activity today are related to computer systems based on software. Software development is a knowledge-intensive activity that requires a detailed study of the subject area and a full understanding of the goals of the product being developed.

A key factor in ensuring the effective use of software and one of the main requirements of users and stakeholders to modern software is to achieve high values of its quality. Software quality is a key factor for its successful implementation and operation. The need to ensure the quality of software stems from the fact that software errors and failures lead to disasters, which lead to human casualties, environmental cataclysms, significant time losses and financial losses.

According to the standards ISO 25010 [31], ISO 25030 [48], SWEBOK [49], we will consider the quality of the software as the ability of software to meet the stated and anticipated needs when using it under certain conditions. The definition of quality in ISO 9000 [50] and ISO 9001 [51] concerns the satisfaction of requirements, i.e. quality in [50, 51] is considered as a characteristic of the software that reflects the degree of its compliance with the requirements. Determining quality from standards [50, 51] does not take into account the fact that requirements may not fully reflect customer needs, then meeting the requirements will not mean meeting customer needs, so such software cannot be considered quality (in fact, this is only formal quality satisfaction).

The chaotic period of software development, when much attention was paid to the program code, rather than its quality, began to recede. In recent years, the software industry has reached a level of development at which software quality assurance requirements have become a mandatory part of software development agreements, as software quality is its most important characteristic from the point of view of stakeholders [52].

In the field of software quality assurance, there are still problems that were noticeable more than 50 years ago – large projects are executed behind schedule or in excess of cost estimates, the developed product does not have the necessary functionality, its performance is often low, software quality does not suit consumers [15-30]. Analytical research and software reviews conducted over the past few years by leading foreign analysts to confirm these not-so-encouraging results. Thus, with a number of methods and tools, the involvement of the best specialists to develop technologies and standards for quality assurance of software, the quality of software still depends on the knowledge and experience of developers [53-58].

Analysis of statistics of success of software projects for 1994-2019, according to The Standish Group International (Chaos reports) [17-26], is presented in Fig.1.1. Successful are projects that were implemented on time, within the budget, with the necessary capabilities

and functions. Projects that were overdue, overspent or did not have the necessary capabilities and functions are challenged. Projects that have been cancelled before completion or have been delivered but are never used are failed.

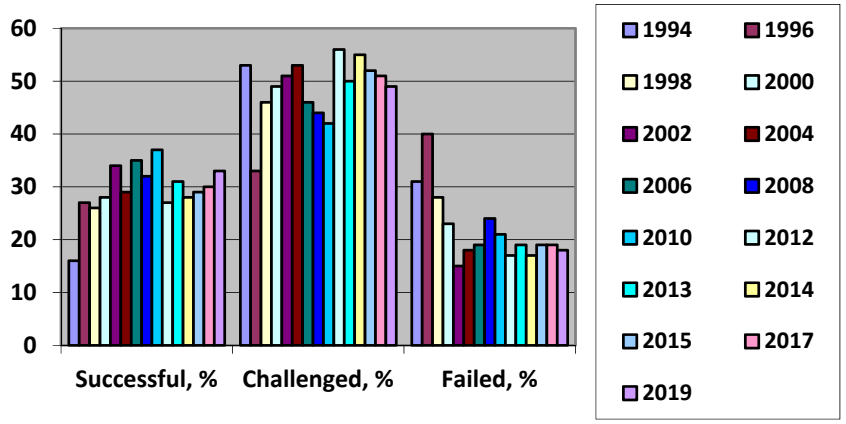

Fig. 1.1. Statistics of success of software projects in 1994-2019

Analysis of data of Fig. 1.1 made it possible to see that the number of challenged projects is quite constant and in recent years is about 50% of software projects.

To date, software projects success rates are as follows – Fig. 1.2 [24].

As can be seen from Fig. 1.2, 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 delays, budget overruns or lack the necessary features.

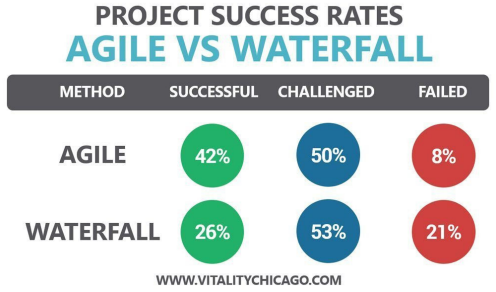

Fig. 1.2. Software projects success rates in 2019 [24]

Research of The Standish Group International shows that 31.1% of software projects will be cancelled until they are completed, and 52.7% of projects will cost 189% of their initial estimates. But such overspending is just the tip of the iceberg. Alternative costs cannot always be measured, but they can reach trillions of dollars. For example, the lack of reliable baggage handling software at Denver Airport costs the city $ 1.1 million per day [18-21].

On average, only 16-29% of software projects are implemented within the planned time and budget (for large companies, this is only 9-16% of projects). In addition, projects implemented by the largest American companies have only about 42% of the required capabilities and functions (for small companies, 78.4% of software projects have at least 74.2% of the initial capabilities and functions) [18-21] – Fig. 1.3.

	CHAMPIONS	UNDER-PERFORMERS
Average percentage of projects completed on time	88%	24%
Average percentage of projects completed within budget	90%	25%
Average percentage of projects that meet original goals/business intent	92%	33%
Average percentage of projects experiencing scope creep	28%	68%
Average percentage of projects deemed failures	6%	24%
Average percentage of budget lost when a project fails	14%	46%

Fig. 1.3. Project Performance Averages of Champions (organizations with 80% or more of projects being completed on time and on budget, and meeting original goals and business intent, and having high benefits realization maturity) versus Underperformers (organizations with 60% or fewer of projects being completed on time and on budget, and meeting original goals and business intent, and having low benefits realization maturity) [22]

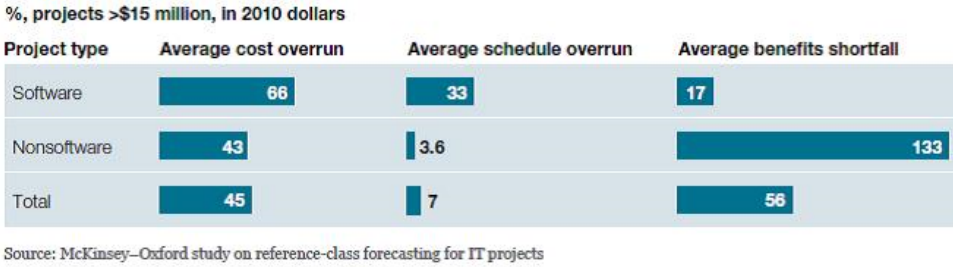

Fig. 1.4. The performance of different types of IT projects varies significantly [59]

Research of McKinsey & Company [9] in collaboration with the University of Oxford also found that half of the large software projects with a total budget of more than $ 15 million

significantly exceeded planned costs, in particular, the average excess of project costs is 66%, average excess of project implementation time is 33%, and the average number of profit losses is 17% (Fig. 1.4, 1.5 [59]).

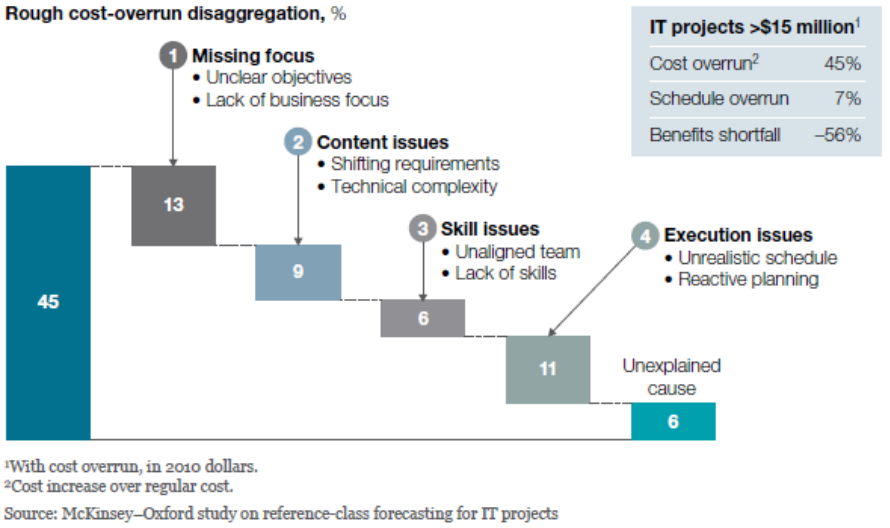

Fig. 1.5. IT executives identify 4 groups of issues that cause most project failures [59]

In general, the nature of software has changed significantly in recent years. The emphasis has shifted from the production of autonomous software products to the production of a software system as a broad system of systems, integrated from a large number of components (subsystems) with interfaces between them [60, 61]. As software becomes more complex and large, the development of a software system will play an increasingly important role in the activities of the developer [56, 62].

One of the most important reasons for the unsatisfactory quality of large software projects, researchers at The Standish Group International call the increase in the number of components (subsystems) and interfaces between them, as well as the uncontrolled complexity of the software system [19-21]. Studies of The Standish Group International (CHAOS reports) [19-21] show that the success statistics of small and large software projects are significantly different (Fig. 1.6).

The analysis of Fig.1.6 provides to conclude that among small projects 62% are successful while among large projects only 6% are successful, and among grand projects only 2% are successful, i.e. small projects in dozens of times are more successful than large projects. This conclusion is confirmed by project statistics based on the use of functional points as the main units of software size [16, 34] – Fig.1.7.

CHAOS RESOLUTION BY PROJECT SIZE

	SUCCESSFUL	CHALLENGED	FAILED
Grand	2%	7%	17%
Large	6%	17%	24%
Medium	9%	26%	31%
Moderate	21%	32%	17%
Small	62%	16%	11%
TOTAL	100%	100%	100%

The resolution of all software projects by size from FY2011–2015 within the new CHAOS database.

Fig. 1.6. Statistics of success of small, moderate, medium, large and grand software projects in 2011-2015 [19]

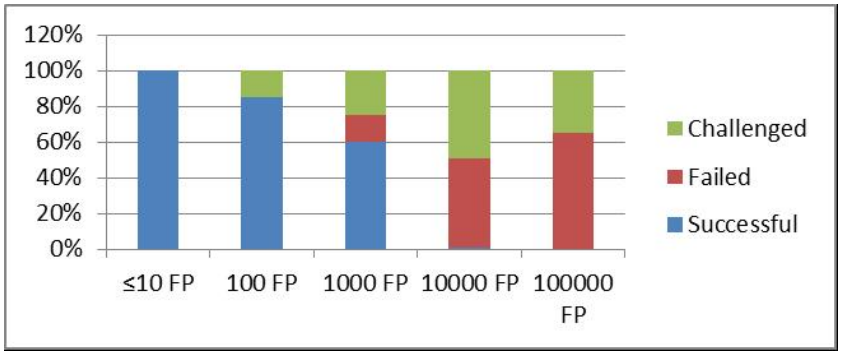

Fig. 1.7. Success statistics of software projects with 10, 100, 1000, 10000, 100000 function points

A significant number of errors are made in the software at the stage of requirements collection [22, 63-66]. Errors of the formation and formulation of requirements and design of the architecture account for 25-55% of all errors, and the greater the amount of software, the more errors are made in the early stages [15]. Statistics show that inaccurate requirements gathering is one the primary causes of software failures (рис. 1.8) [22, 65]. Ideally, all problems encountered during the requirements phase should be resolved before the design starts. The late discovery of requirements errors is the most expensive to correct [67].

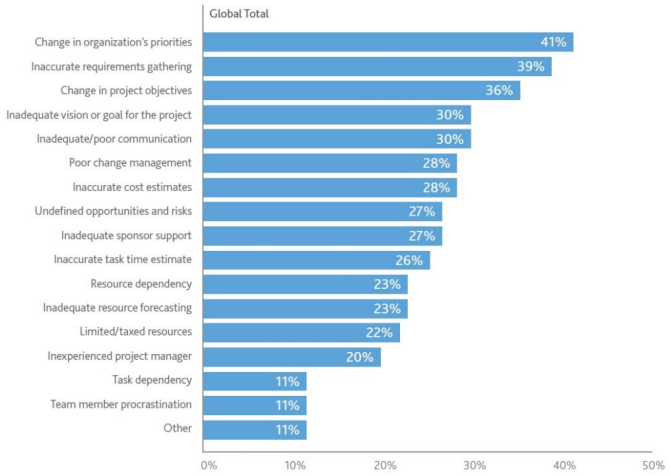

Fig. 1.8. The primary causes of software failures [22]

Software project success factors are represented on Fig. 1.9 [25]. Obviously that clear requirements has 13% of weight in project success factors.

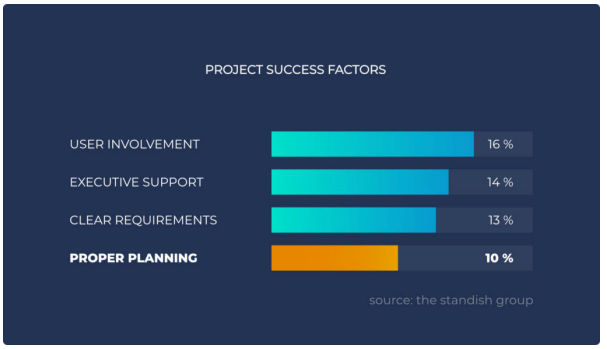

Fig. 1.9. Software project success factors for 2020 [25]

The prime cause of the software failures and crashes is poor design documentation, especially at the system level [30, 68] – Fig. 1.10. Poor documentation causes many errors and reduces efficiency in every phase of a software product's development and use, leads to the failure or challenges of the software project [23].

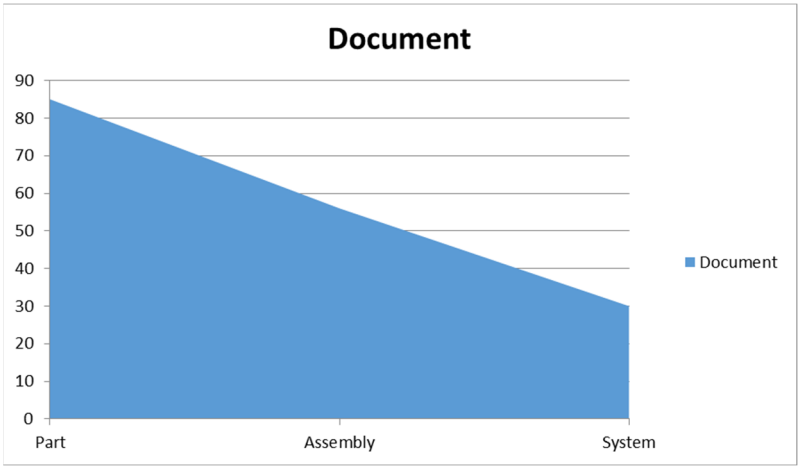

Fig. 1.10. Percentage of documentation of knowledge at different stages of software systems development

Papers [27, 69] confirms the fact that the causes of almost all software incidents and disasters lie in the specification of software requirements. The vast majority of software-related accidents are caused by erroneous requirements rather than coding errors. Paper [69] describes the results of an experiment conducted to confirm or reject the hypothesis that software failures and errors written by different developers according to the same specification are statistically independent. During the experiment, several independent development teams wrote their own version of the software according to one specification. As a result of this experiment, it was found that versions of software written by different developers on the same requirements contained a number of common errors related to errors or inaccuracies in the requirements (specifications).

It is desirable to identify and eliminate defects in the requirements before they begin to affect later stages of development. Early stages of the life cycle affect the quality of software more than late stages, so the time spent on quality control in the early stages provides an opportunity to reduce defects, reduce development time and reduce costs at later stages [55].

The earlier the defect (error, violation, malfunction) is detected, the cheaper it will be to correct it. As shown in Figure 1.11, the cost of correcting defects detected after the release of the product, almost 100 times higher than the cost of correction, if the shortcomings were identified in the process of forming and formulating requirements [30].

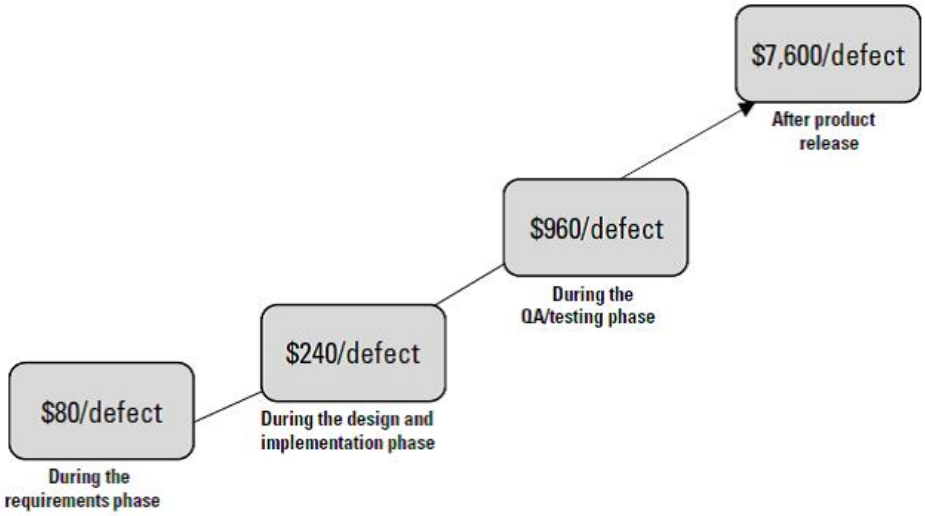

Fig. 1.11. The cost of correcting defects increases dramatically through the development process [30]

If you consider alternative design options and analyze the impact of errors in the requirements before the creation of the system as such, it will provide significant savings on error correction (Fig. 1.12). And if the study of costs usually takes hours or even days, then, in this case, will go for minutes [30].

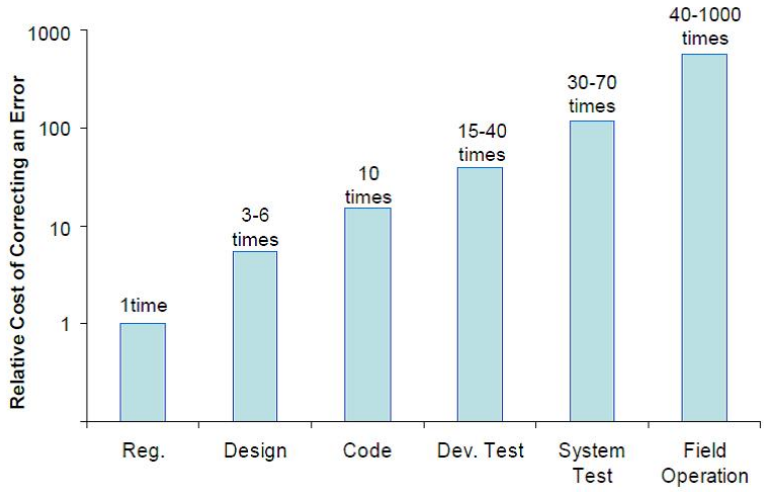

Fig. 1.12. The time spent on error correction during the software life cycle [30]

As the interval between the time of introduction and detection of the defect increases, the cost of its correction increases sharply. The longer the error persists in the software development chain, the more it penetrates other parts of the software, the more damage it causes in the following stages, and the more money will have to be spent on its elimination.

Dozens of companies have found that a policy of early correction of defects can reduce several times the financial and time costs of software development, which is a strong argument in favour of the earliest possible detection and elimination of defects. Studies of 50 NASA Software Design Laboratory projects have shown that increased attention to early quality control can significantly reduce the level of errors, but does not increase the overall cost of development [15].

In the process of formulating and formulating requirements, information losses may occur due to incomplete and different understanding of information needs and context – especially such losses are significant for software projects that are developed at the intersection of subject areas when it's necessary to consider standards for software development and standards for subject areas, for which software is being developed. Such a number of standards is difficult to implement, and even more difficult to assess the extent to which the recommendations of these standards have been taken into account.

The result of software production as a system of systems with a large number of components with interfaces between them is the fact that incidents and accidents related to software today contain a new type of accident caused by component interactions: each component does not contain errors (meets needs of the customer), but incorrect interactions between components lead to problems. Component interaction accidents are the causes of major accidents due to the increasing complexity of software systems, which leads to the impossibility of predicting all possible consequences of interactions between components [27, 69].

Simple improving the quality and reliability of individual components of software systems cannot prevent such accidents, because they are not caused by the failure of individual components. The high quality and reliability of the components do not prevent such accidents. These problems are exacerbated by the fact that reliability and safety engineering methods such as FTA and FMEA, which were developed to detect component failures, cannot be used effectively to prevent component interaction failures. The problem faced by engineers is that they do not have other, more suitable, tools to determine the system properties of the software, as well as to predict interaction accidents [27, 69].

The quality of software may be low due to the fact that insufficient attention is paid to the issue of attitude to the information of the subject area at different stages of the software life cycle, its sufficiency, reliability, refinement. Some information is analyzed too meticulously, and some are not taken into account at all. Information of the subject area with low probability is often rejected, and sometimes its probability is not estimated at all. New information can come at different stages of the life cycle – both at the stages of formation and formulation of requirements and design of architecture, and at the stages of implementation and operation, but it is often neglected. Such neglect of subject area

information at all stages of the life cycle is one of the critical problems in software development [70-72, 56, 62].

Fig. 1.13 shows a situation characterized by the prematurity of design decisions and their documentation to understand the design, emergent properties and their impact on the functional characteristics of the product. This area is called the "knowledge gap", the presence of which is a practical result and the root cause of many engineering failures [73].

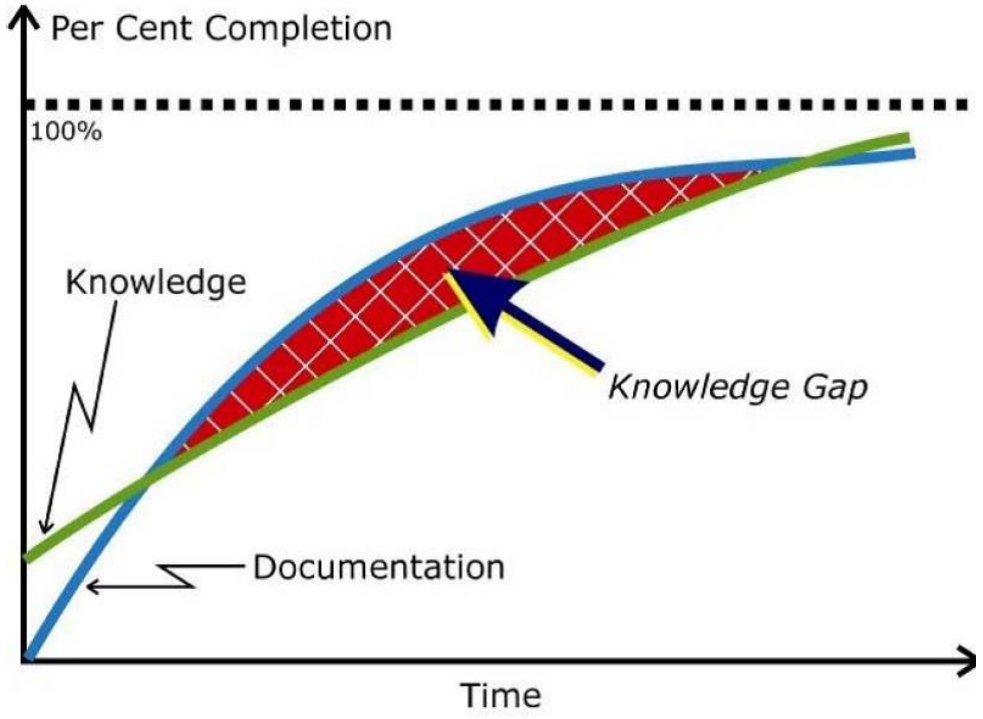

Fig. 1.13. Knowledge Gap [73]

The fact that the information of the subject area is partially ignored at different stages of the software life cycle indicates that the size of the knowledge gap is not constant for the software project – in the life cycle, it can increase and decrease as new information appears.

For software engineering, the point of view on the knowledge gap presented in Fig.1.13 does not quite correspond to reality. Partial consideration of subject area information in software quality assessment models and its impact on the finished product leads to an increase in the size of the knowledge gap in the life cycle, which can cause failures and other software problems.

Given the above, all available knowledge and information about the software system will be presented in the form of a diagram, which has a sector that shows the amount of insufficient (unknown) information (knowledge gap) – Fig. 1.14.

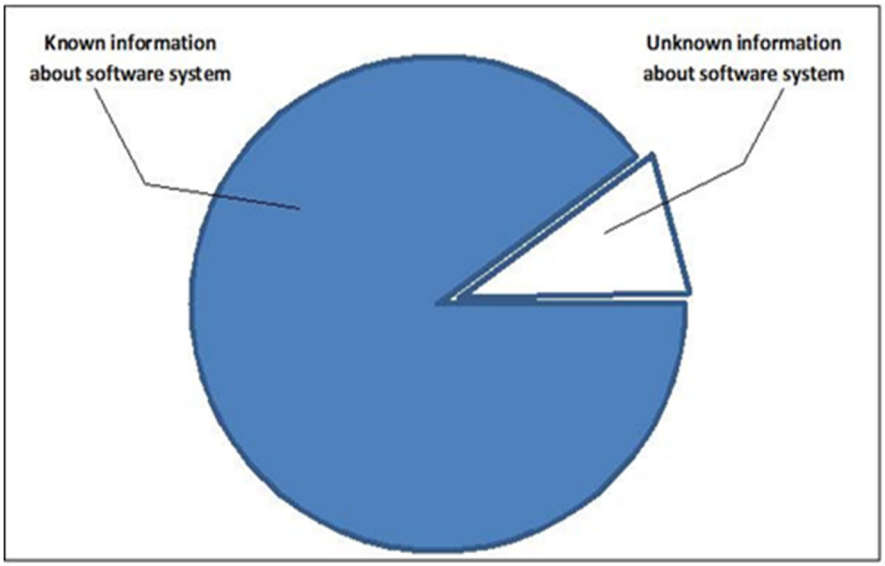

Fig. 1.14. Field of knowledge about software system with the sector of unknown (or unconsidered) information

This sector consists of unconsidered information of the subject area (including information missing in the specification of software requirements). In the process of working on a software project, it is important to assess the share of information uncertainty of the project. The size of the sector with unknown information has not been determined, as it is unclear which and how much information remains unknown. The sector with unknown information should be narrowed – by taking better account of the information of the subject area, starting from the early stages of the life cycle. The smaller the sector of unknown information, the better and more quality the software system and the more secure it will be. Therefore, the approach to reducing the share of unknown information about the software system is actual. This can be achieved by more fully taking into account (reducing losses) information of the subject area when evaluating the information on the quality of software.

From the definition of software quality as the degree of user satisfaction or the degree of conformity of software to customer needs, it follows that if the project objectives set in the early stages of the life cycle do not meet user needs, the software cannot be high-quality, even if modern technologies and the most qualified developers were involved. Therefore, the quality and success of the software project significantly depend on the specification of

requirements, as well as on the sufficiency of the information available in requirements (as the presence of all information elements (data), which are necessary to determine the quality of the software). Such experimental evidence directly leads to the need to deepen the analysis of the specification [55, 57, 58].

Definition 1.1. Sufficiency of information on quality in the specification of software requirements is the presence in the specification of all information elements, which necessary to determine the software quality [74-76].

Assessing the sufficiency of software requirements specification's information provides the ability to choose a software project from the standpoint of its projected quality, increases the efficiency of project management due to the validity of decisions, reduces the time for their development and adoption, and reduces the cost of collecting and processing information. Insufficient information of the software requirements reduces the effectiveness and veracity of software quality assessment.

The software requirements determine the required software quality characteristics, as well as influence the methods of quantitative evaluation and the acceptance criteria formulated for the evaluation of these characteristics. Therefore, all the necessary information on quality is already contained in the specification of software requirements, i.e. on the basis of the specification of software requirements, it is possible to assess the sufficiency of information for further determination of software quality. If some quality information elements are missing, the requirements specification does not have enough information to determine the quality of the software, and developers must make the necessary additions to the specification.

Thus, as shown by the analysis of the impact of requirements' information on software quality, the quality factors of modern software systems are less dependent on the writing of software code, but significantly depend on the formation and formulation of requirements and architecture design. Defects introduced at the stages of formulating and formulating requirements and designing the architecture should be identified and eliminated before they begin to affect the results of later stages of the life cycle. The quality and success of the software project significantly depend on the specification of software requirements.

Under such conditions, the analysis of the specification of software requirements, the ability to "cut off" software projects with incomplete (with insufficient information) specification is actual and very important. Sufficiency of information is one of the most important aspects of software quality assessment.

1.2. Review of Information Technologies, Tools for Software Requirements Analysis and for Assessing the Sufficiency of SRS Information

Today the following approaches for assessing the sufficiency of information of requirements to the software are known – Table 1.1 [77].

The review of the known information technologies and tools for analyzing the software requirements is given in Table 1.2 [77].

Tab. 1.1. The review of approaches to assessing the sufficiency of information of requirements to software [77]

The approach	Limitations of the approach
Model for validation of sufficiency of safety requirements, focusing on sufficiency of hazard identification, hazard analysis, and software safety requirements traceability [78, 79]: several different metrics have been introduced, in particular, Percent Software Safety Requirements – as the ratio of the number of software safety requirements to the total number of software requirements; the authors suggest comparing this metric with a similar metric for implemented software, on the basis of this comparing the conclusion about the sufficiency of software safety requirements is made	On the basis of the proposed metric, only the sufficiency of the number of safety requirements can be assessed, but not the sufficiency of their information; the impossibility of interpretation by comparing the proposed metric for fundamentally new software; there is no tool for automatically identifying and calculating the safety requirements in the software requirements specification
Evaluation of testing sufficiency as the achievement of the test coverage levels recommended or mandated by safety standards and industry guidelines [80]	The approach is aimed only at verification of software and requirements, but not at the validation of the developed software and customer needs; the approach uses the SRS solely as an input for the developed tool, but doesn't check the requirements of the specification for their sufficiency

Tab. 1.2. The review of approaches to assessing the sufficiency of information of requirements to software [77]

Information technology (IT) or tool	Limitations of the IT or tool
CORE: enables the user to extract the requirements from the source documenta-tion and then analyzes them for completeness, consistency and testability [81]	Checks the completeness of the requirements with respect to business requirements, but doesn't provide to check business requirements for their completeness

Visure – checker of requirements quality using natural language processing and semantic analysis [82]	Doesn't provide validation of compliance with the requirements of the requirements to the needs of the customer
Accompa: provides automatic requirements gathering; automatically detects & track dependencies between requirements [82]	Commercial tool – costs at just $199/month with no installation and maintenance [83]; doesn't check if all needs of the user have been reflected in the requirements
Innoslate: analyzes requirements using natural language processing technology [82]	Commercial tool – Innoslate Cloud is priced at $49/user/month and Innoslate Enterprise – $199/user/month [83]; doesn't provide quantitative assessments of the properties of requirements information
ReqView: organizes requirements into a tree hierarchy, uses rich text format for requirements description [82]	Doesn't verify and validate the requirements of the specifications for the needs of the customers
Modern Requirements4TFS: runs traceability analyses to ensure quality and find gaps or dependencies in requirements [82]	Doesn't identify the needs of the user that were not reflected in the requirements; doesn't provide visualization of the found gaps in the requirements
Natural language processing (NLP) Requirements Analysis Tools (for example, QVscribe): automate and significantly speed the searching the possible errors in natural language requirements [29]	Doesn't reveal information losses in the formation of requirements
Requirements Analysis Tool: uses of user-defined glossaries to extract structured content; Semantic Web technologies are leveraged for deeper semantic analysis of the extracted structured content to find various kinds of problems in requirements documents [84]	Limitation of the glossary on which the tool is based

QARCC (Quality Attribute Risk and Conflict Consultant): is the 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 [85]	Doesn't check the sufficiency of SRS requirements (in particular, non-functional requirements)
Requirements Assistant: identifies the missing requirements and inconsistency in requirements; detect the lack of some type of non-functional requirement such as reliability, security, safety [86]	Commercial tool; doesn't provide quantitative assessments (metrics) about missing non-functional requirements
QuARS Requirements Analysis Tool: provides screening of the requirements on consistency, completeness; identifies 37% of requirements defects [85, 87]	Commercial tool
DESIRe: ensures that the rules of completeness, non-ambiguity and comprehensibility are respected [88]	Commercial tool
RQV Tool (Requirement Quality Verification Tool) [88]: a semi-automatic verification tool based on a comprehensive quality model	Provides assessments of the quality of requirements information, but not its completeness or sufficiency

The review of known approaches for assessing the sufficiency of information of requirements, and information technologies and tools for the SRS analysis has shown, that there are a number of effective solutions, but they all don't solve the problem of assessing the sufficiency of quality information in the specification of software requirements. In addition, they belong to different methodological approaches, are designed to different tasks and don't integrate one with one. So, there is currently no information technology for assessing the sufficiency of information at the initial stages of the software life cycle.

1.3. Two Approaches to Software Quality Assessment

Today, two approaches to software quality assessment are key – according to the SQuaRE model (ISO 25010:2011 standard [31]) and based on the results of the metric analysis.

The general approach to software quality assessment according to the SQuaRE model (ISO 25010:2011 standard [31]) is to first identify a small set of quality characteristics of the highest level of abstraction and then in the direction of "top-down" to divide these characteristics into sub-characteristics and sets subordinate attributes.

According to the standard [31], a characteristic is a set of software properties that are used to describe and evaluate its quality. Characteristics are the basis for the formation of software requirements. Software quality characteristics can be refined on the basis of complex indicators (sub-characteristics), which are based on the ability to meet the stated or emerging needs. The software quality sub-characteristic is expressed by weighted arithmetic mean, taking into account the values of the attributes that evaluate this sub-characteristic and their weighted factors. In turn, an attribute is a physical or abstract property of the software that can be measured.

The quality model created within the standard [31] is determined by 8 general characteristics of product quality: Functional Suitability, Performance Efficiency, Compatibility, Usability, Reliability, Security, Maintainability, Portability – Fig. 1.15.

Fig. 1.15. Software quality model according to the standard ISO/IEC 25010:2011

Each software quality characteristic is a function of several sub-characteristics of quality (total of 31 sub-characteristics, according to the standard [31]). The lower level of the hierarchy is represented by software quality attributes (measures) that are subject to accurate description and measurement. Software quality attributes are defined and described in ISO

25023:2016 [32]. The analysis of papers [89-92] and the standard [32] made it possible to determine the dependence of quality sub-characteristics on 203 attributes (measures).

The concept of software quality assessment is presented in Fig. 1.16.

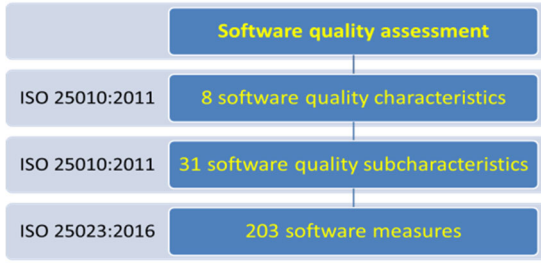

Fig. 1.16. The modern concept of software quality assessment by ISO 25010:2011

To date, the evaluation of software quality characteristics in accordance with ISO 25010 [31] is as follows. First, the software quality attributes are evaluated – the evaluation scale is graduated depending on the possible degrees of compliance of the attribute with the imposed restrictions. Software quality sub-characteristics are evaluated on the basis of a set of attributes, and software quality characteristics are evaluated on the basis of a set of sub-characteristics. But the evaluation of software quality attributes is performed subjectively, as there are no uniform standards for their evaluation. Interpretation of attribute values is also subjective, as there are no standardized "reference" attribute values. Graduation of the rating scale is again subjective because it depends on the possible degrees of compliance of the attribute with the imposed restrictions, and the degrees of compliance are not standardized and are determined by the software organization [15, 16].

Therefore, the evaluation of software quality as a function of the main eight characteristics is subjective, because the software organization interprets the obtained values of attributes as maximum, scales the scale of evaluation of each characteristic, based on its own interpretation of attribute values and possible degrees of matching attributes to constraints, resulting in maximum values of each characteristic, and accordingly the maximum value of software quality. In fact, there is only a formal satisfaction of the quality of the software due to incomplete coverage of the standards of standardization objects, as well as due to the choice by the developer of favourable standards and adaptation of these standards to their needs.

The main idea of the SQuaRE model [31] is that the evaluation of software quality, as well as its characteristics and sub-characteristics, should be carried out comprehensively, taking into account all these characteristics, sub-characteristics and attributes, respectively. But there are no comprehensive methodologies that will allow assessing not only the impact of each individual characteristic on the quality of software but also provide an opportunity to

assess the availability of all attributes needed to determine all quality sub-characteristics and characteristics (sufficiency of information) and to evaluate the interaction of characteristics.

Analysis of papers [89-92] and standards ISO 25010 [31], ISO 25023 [32] made it possible to conclude that there are attributes on which depend more than one sub-characteristic and characteristics of software quality, i.e. there is a correlation of sub-characteristics and characteristics by certain attributes (according to the standard [32] sub-characteristics of quality depend on 203 attributes, but only on 138 different attributes). Fig. 1.17 shows an example of the correlation of sub-characteristics and quality characteristics of software by the attribute "Operation Time" [93, 94].

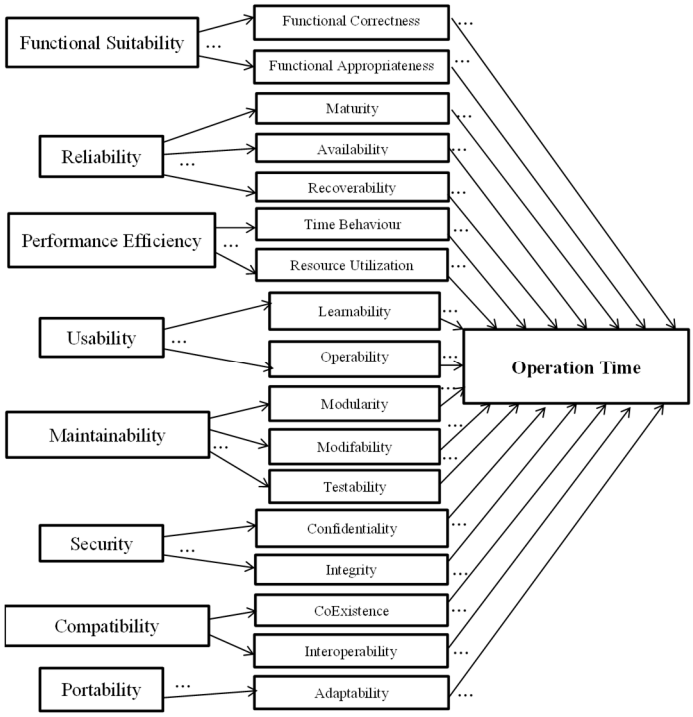

Fig. 1.17. Correlation of software quality sub-characteristics and characteristics by measure "Operation Time"

Fig. 1.17 shows that, for example, 17 (out of 31) sub-characteristics of software quality and, accordingly, all 8 characteristics of software quality depends on the attribute "Operation Time".

When evaluating the quality of software, to eliminate the problem of subjective evaluation and formal quality satisfaction, it is necessary to take into account both the possibility of calculation and the degree of expression of characteristics, sub-characteristics

and quality attributes, and their significance. The existence of relationships (correlations) between the characteristics and sub-characteristics of the attributes, shown in Fig.1.17 on the example of the attribute "Operating time", affects the significance and weight of the attributes of software quality [93-95]. If the attributes that are part of several sub-characteristics and (or) quality characteristics are defined inaccurately or absent (there is an insufficiency of information), the simultaneous use of these attributes will significantly affect the veracity of the obtained software quality estimates. In such a situation, it is important to mitigate the effect of cross-correlation of such characteristics and sub-characteristics when using them in the quality model. This mitigation is accomplished by identifying joint attributes, ensuring their presence, increasing the accuracy of their values, or, if possible, limiting the simultaneous use of sets of sub-characteristics that contain the same attributes.

Interactions of sub-characteristics of quality on joint attributes are presented in Fig. 1.18. The identified interactions (by joint attributes) of quality characteristics are presented in Fig. 1.19.

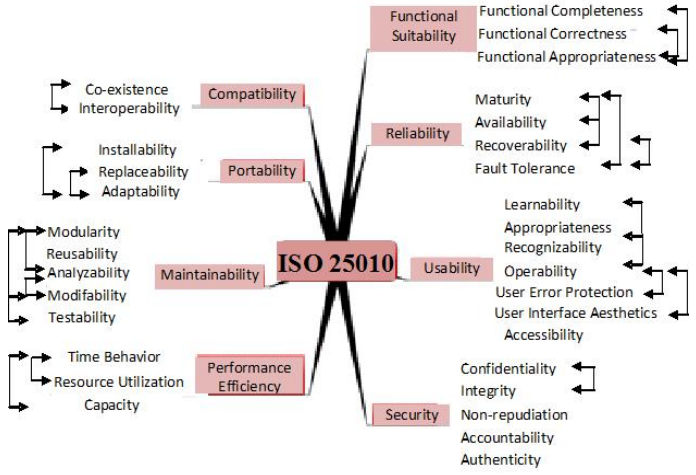

Fig. 1.18. Interactions (by joint attributes) of software quality sub-characteristics (within quality characteristics)

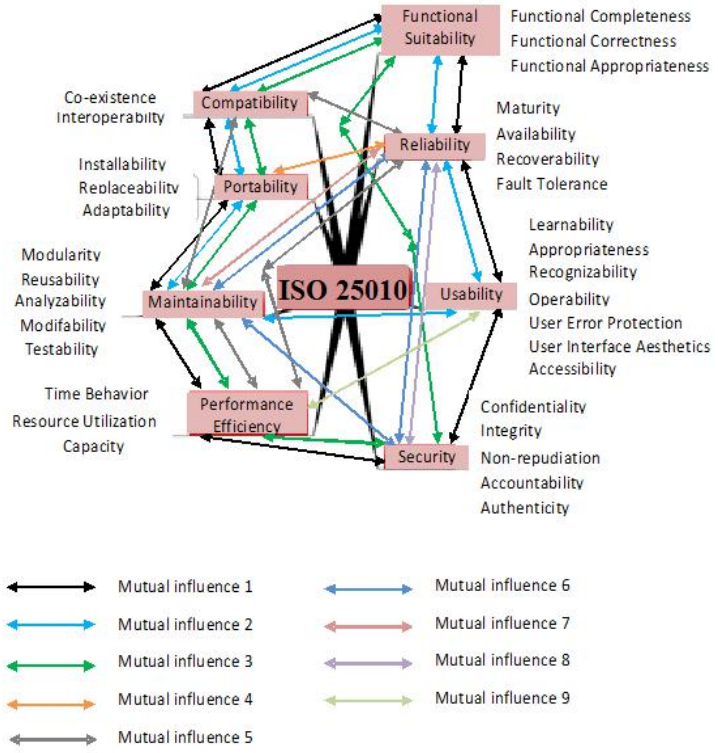

Fig. 1.19. Interactions of software quality characteristics

In [93, 94] a method for estimating the weights of software quality attributes was developed. Using the developed method, the evaluation of the weight coefficients of the software quality attributes was performed. Attribute weights used to calculate more than one sub-characteristic are calculated according to the developed method and are presented in Table 1.3.

Since all other attributes are used in the calculation of only one sub-characteristic of quality, the weights of the other attributes are equal to 1/138. The numerator of the weighting factor of each quality attribute indicates the number of software quality sub-characteristics that depend on this attribute.

Tab. 1.3. Weights of joint measures [93]

Measure	Weight	Measure	Weight
Number of Functions	11/138	Functional Implementation Coverage	2/138
Functional Implementation Completeness	2/138	Operation Time	17/138
Functional Sufficiency	2/138	Precision	2/138
Number of Data Items	8/138	Number of Tasks	2/138
Number of Failures	6/138	Number of IO-Related Errors	2/138
Number of Test Cases	5/138	Number of Instances of Data Corruption	2/138
Number of Breakdowns	2/138	Number of Access Types	2/138
Number of Faults	3/138	Access Controllability	2/138
Number of Resolved Failures	4/138	Number of Controllability Requirements	2/138
Number of Illegal Operations	3/138	Number of Evaluations	2/138
Number of User Errors or Changes	2/138	Mean Amount of Throughput	2/138
Number of Interface Elements	2/138	Error Time	2/138

The second approach to software quality assessment is a quality assessment based on the results of the metric analysis. The modern software industry has accumulated a large number of metrics that evaluate individual production and operational properties of software.

According to ISO 24765 [96], a metric is defined as a measure of the degree of ownership of a property that has a numerical value. In general, a software metric is a measure that assigns a numerical value to a software property as weighted arithmetic mean, taking into account the values of the indicators that evaluate this metric and their weights.

As a result of the analysis of complexity and quality metrics in terms of the possibility of their application in the early stages of the software life cycle to obtain an accurate or predicted value, a number of metrics were identified that can be used at the design stage – 10 metrics with accurate values at the design stage and 14 metrics with the predicted values at the design stage [97-99].

Today, the evaluation of software quality and complexity based on the use of metric analysis results is as follows (Fig. 1.20) – based on quality and complexity indicators, metric values are calculated, which, in turn, provide a comprehensive assessment of software quality and complexity.

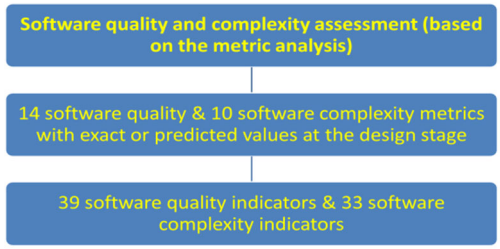

Fig. 1.20. The modern concept of assessment of software quality and complexity based on the metric analysis

The analysis of papers [37, 100] made it possible to determine the dependence of software quality and complexity metrics on 72 indicators, but only on 42 different indicators. Metrics that have the joint indicator(s) are mutually correlated.

Fig. 1.21 shows an example of the correlation of software quality and complexity metrics on the indicator "Quantity Of Modules" [93].

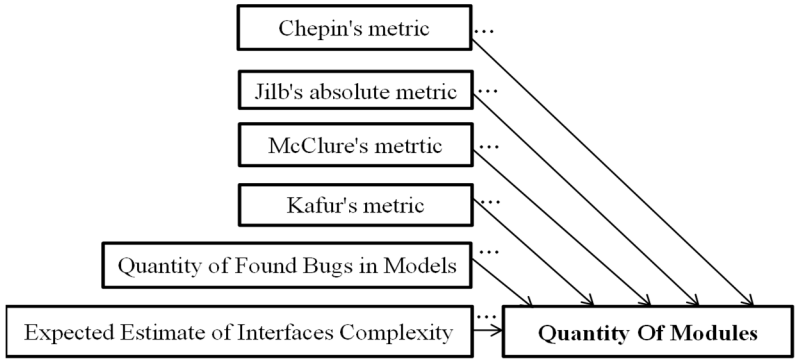

Fig. 1.21. Correlation of software quality metrics by measure “Quantity Of Modules”

Fig.1.21 shows that, for example, 6 (out of 24) metrics of complexity and quality of software and, respectively, all 3 (from 4) characteristics of the software (project complexity, project quality and predicted software complexity) depend on the indicator "Quantity Of Modules".

If the value of the indicator(s), by which the metrics are correlated, is inaccurate, then, respectively, the degree of veracity of all metrics will decrease. If the value of this indicator(s) is absent in the specification of software requirements, the possibility of calculating several metrics will disappear (for example, the absence of the indicator "Number of modules" leads to the impossibility of calculating six metrics). Thus, the inaccuracy of the definition or the absence of indicators (that are part of several metrics) in the specification of software requirements will significantly reduce the veracity of the obtained estimates of the complexity and quality of the software project and software.

In such a situation, it is important to mitigate the effect of cross-correlation of such metrics when using them in models or comprehensive assessments of complexity and quality. This mitigation is accomplished by identifying joint indicators, ensuring their availability, increasing the accuracy of their values, or, if it's possible, limiting the simultaneous use of metric sets that contain the same metrics.

Using the method of estimating the weight coefficients of software quality attributes developed in [93, 94], the estimation of weight coefficients of software quality indicators was performed. The weights of the indicators, used to calculate more than one metric, are presented in Table 1.4. Since all other indicators are used in the calculation of only one metric, the weights of the remaining indicators are 1/42. The numerator of the weight of each quality indicator indicates the number of software metrics that depend on this indicator.

Tab. 1.4. Weights of joint indicators [93]

Indicator	**Weight**	**Indicator**	**Weight**
Quantity of Modules	6/42	Total Quantity of Operators	4/42
Quantity of Preceding Modules	2/42	Total Quantity of Operands	2/42
Quantity of Following Modules	2/42	Quantity of Unique Operands	2/42
Quantity of Code Lines	12/42	Cost of One Line	3/42
Project Duration	4/42	Project Type	2/42
Share of Design Stage in Life cycle	2/42		

When assessing the quality of software, for ensuring the appropriate level of reliability, it is important to satisfy the presence in the specification of software requirements those attributes and indicators that have higher weights.

1.4. Ontologies and Ontology-Based Intelligent Agents for the Software Engineering Industry

For increasing the veracity of software quality assessment, the knowledge of specialists who already have experience in quality assessment and complexity for different types of software is of great value. For example, knowledge about the interaction and correlation of characteristics and sub-characteristics of quality by attributes, as well as metrics of complexity and quality by indicators, is valuable because some of the attributes or indicators by which correlation occurs may be omitted altogether, as a result, the accuracy and veracity of the obtained estimates may deteriorate, etc.

Information on software quality assessment according to ISO 25010:2011 (for example, relationships of characteristics and sub-characteristics by attributes), as well as on determining the quality and complexity of software and software based on the results of metric analysis (for example, relationships of metrics by indicators) it is convenient to present in the form of ontologies that allow displaying causal relationships between concepts.

An ontology is a specification of a conceptualization, where conceptualization is a description of concepts, as well as all the information that is relevant to the concepts and necessary to describe and solve the problems of the subject area. Ontologies are used to reflect known knowledge, as well as the acquisition, structuring of knowledge and the formation of new knowledge of the subject area.

Ontologies are formal concepts of specific subject areas. They allow you to conceptualize a domain by capturing entities and relationships in the domain. Determining the relationships, in which an entity is involved, partially allow to understand its meaning (content), as it provides an opportunity to see where the entity is in a relationship with another domain.

Ontology is a collection of concepts which able to model terms of vocabulary into a domain knowledge [101]. Ontology provides a better understanding of contextual knowledge. From the perspective of computational science, ontology is defined as a concept to model the system structure. For example, the relevant entities and relationships that exist from observations are useful for specific purposes. Ontology generally has fundamental form components, i.e., class, instance, and relation [102]. Ontology is flexible, easy to modify, understood by humans and machines, and able to integrate with machine learning. Ontology is associated with discovering and modeling reality under particular perspectives. It focuses on the structure and nature of an object. Ontology is also referred to a representational knowledge. Ontologies can describe abstract things (work processes, knowledge or tasks) as well as real things (devices).

The advantages of using ontologies are a systematic approach to the study of the subject area, the ability to holistically present known information of the subject area, identify duplications and gaps in knowledge based on visualization of missing logical connections, the ability to access, understand and analyze information by intelligent and non-intelligent agents (which is very actual in the modern transition to the era of the Semantic web, when resources must be understandable not only to humans but also to agents) [41, 103].

Ontologies are defined as a key technology for the development of the semantic web and play a critical role in the organization of information processing on the basis of the Web, its sharing and its exchange between applications.

Formally, the ontology is defined as: $O =< X, RX, F >$, where X – a finite set of concepts of the subject area, RX – a finite set of relations between concepts, F – a finite set of interpretation functions given on concepts or relations.

The idea of using ontologies as a means of integrating data, knowledge and requirements for software is not new in itself. Thus, weighed ontologies were used by scientists to trace software requirements [42, 43, 104, 105]. The aim of [106] is the use of a domain ontology for software analysis and reengineering tools. The paper [47] presents an ontological model for describing and defining the subject and operational knowledge to ensure the quality of software.

Methods and tools for developing the software systems based on tasks' ontological models are proposed in [41, 103]. The authors of [41, 103] propose to use ontological models at all stages of the software life cycle by conceptualizing the subject area in the context of the tasks solving. That is, in the subject area the necessary entities are identified, their attributes and limitations, dependencies between them are determined and an ontology of the subject area is developed, which can be further used by developers. The authors [41, 103] proved that the use of such an ontology of the subject area simplifies the process of initial conceptualization when creating software, avoids conceptualization errors in the early stages of the software life cycle. But the use of such an approach is impossible if the software is developed for a subject area for which an ontology has not yet been developed. In addition, the software development should take into account the requirements and standards not only of the subject area but also the standards for software development, which is in no way taken into account in the approach described in [41, 103]. Also, methods and tools for developing the software systems based on tasks' ontological models do not allow to automatically analyze the specifications of software requirements, in particular, for the sufficiency of their information, although it is automated knowledge processing ensures minimization of information loss.

The authors of [43] propose an ontological structure of the traceability of multi-purpose requirements MUPRET for processing the heterogeneity of software requirements based on the automatic generation of traceability relations. The accuracy of the traceability relations, which are generated by the MUPRET structure is checked by comparison with a set of traceability relations that are manually identified by users. In [105], weighted

ontologies are used to process natural language in the transition from the specification of requirements written in natural language to software design. The ontologies, as the authors [105] argue, are able to show the inconsistencies of the requirements, which are presented in natural language. The presented approaches are aimed at eliminating the heterogeneity or inconsistency of the existing requirements in the specifications, but in no way verify whether all the needs and requirements of the user were reflected in the specifications. Therefore, they are not suitable for assessing the sufficiency of information in the software requirements specifications.

An ontological model for describing and defining the subject and operational knowledge to ensure software quality was developed in [47]. The authors of [47] proposed an ontology that combines knowledge about the terminology and semantic relationships of SWEBOK, IEEE, ISO standards aimed at ensuring software quality. The developed ontology simulates the process of the software life cycle and software quality assurance. But quality assurance occurs only at the end of the software life cycle for the finished software code. Today, software quality is interpreted as its ability to meet the needs of the customer when used under certain conditions, so all the necessary information about the customer's needs must be included in the specification of software requirements, i.e. the specification provides the opportunity to assess the sufficiency of information to further achieve software quality. The approach proposed in [47] does not provide for software quality assurance in the early stages of the life cycle, in particular, does not involve assessing the sufficiency of information of quality information in the software requirements.

Another advantage of using ontologies and weighted ontologies is the ability to access, understand and analyze information by intelligent agents (which is very actual in the modern transition to the era of the Semantic web, when resources must be understandable not only to humans but also to agents).

The intelligent agent is a system that observes the environment, interacts with it, and its behaviour is intellectual in the sense that the agent understands the essence of their own actions, and these actions are aimed at achieving a certain goal [107, 108]. Such an agent can be a software system, bot, service. The intelligent agent uses in its operation information obtained from the environment, analyzes it, comparing it with the facts already known to him and, based on the results of the analysis, decides on further action.

A number of studies are devoted to solving the problem of developing ontology-based intelligent agents – Table 1.5.

Tab. 1.5. Known ontology-based intelligent agents for software engineering

Ontology-based intelligent agent for software engineering	**Author(s)**
Ontology-based intelligent agent for eliminating uncertainty in software requirements and improving stakeholder communication	K. Ossowska et al. [109]

Unified methods for developing intelligent agents for planning activity using an ontological approach to improve the efficiency of the processes of functioning of such systems	V. Lytvyn et al. [110]
Multi-agent systems, in which agents make decisions based on knowledge taken from ontologies, which allows integrating the agent platforms with semantic web data and ontologies	A. Freitas et al. [111]
The use of ontologies for agent-oriented software engineering; offers a tool that uses existing ontological constructs to create software code, and also experimentally confirms the benefits of using ontology-based agents for the software engineering industry	A. Freitas et al. [112]
Ontology-based intelligent agent for minimization of the existing semantic uncertainty at the development of the specification of requirements to the software in natural (Spanish) language, for automatic reception of the basic elements of the specification and for automatic construction of the diagram of the purposes	L.A. Lezcano-Rodriguez et al. [113]
Methodological approach to engineering systems based on ontology-based intelligent agents	V. Hilaire et al.[114]
Ontological agent-oriented models for formalizing the initial software requirements in order to reduce costs on the example of developing applications for Ambient Assisted Living for patients with Parkinson's disease	I. Garcia-Magarino et al. [115]
Task-oriented architecture based on agent-oriented paradigm and ontological design for decision support systems (for example, a clinical system for emergency care), which allows iterative transfer of functional requirements into architectural components	S. Wilk et al. [116]
The basis for the formal presentation and verification of requirements and ensuring the functional correctness of resource-constrained contextual systems of critical application in the form of ontology-based intelligent agents	A. Rakib et al. [117]
Approach of concept based software engineering, which is focused on supporting productivity and quality goals during the development of software systems	O. Meyer et al. [118]
To improve the risk management in software projects based on intelligent agents and fuzzy systems	C. De Olveira et al. [119]

The related problems of Agent-oriented programming and puts forward Agent-oriented programming framework, which has important use value in overcoming software crisis	H. Yan [120]
The Agent-Based Intelligent Tutoring System uses Function Point Metrics as the domain knowledge and performs function point analysis in an environment that provides visualization, immediate feedback, interactive and guided help	A. Rahman et al. [121]
Interactive, user-centered agent-based approaches support the designer at the beginning of, and during, conceptual software design (conceptual software engineering design is a difficult task for software engineers to perform)	C. Simons et al. [122]
A multi-layer agent-based approach to address the implementation of the business application design rules throughout the software life cycle. The application framework described in this paper captures and clarifies the key issues of the business application design through the deployment of aspect-orientation and intelligent agents that can learn and adapt to the environmental changes in order to ensure the business application design rules throughout the software life cycle	F. Akkawi et al. [123]
Application cases of intelligent systems technologies in modern product life cycle management (PLM) system. The current PLM software includes intelligent agents which automate the development of the product. In particular, authors propose a new approach, based on the ontology, the Semantic Web and the concept of agent, to automate the development of a new product	V. Karasev et al. [124], A. Abadi et al. [125]
A framework for using software agents during software life cycle. Among other tasks, agents could assist with the activities of software quality assurance, project management, and maintenance. A prototype implementation of the agent concept for testing Java classes has been presented too	T. Philip [126]

The conducted analysis of ontological models, methods and tools for assessing the initial stages of the software life cycle showed that they do not solve the problem of a quantitative assessment of the initial stages of the software life cycle based on specification analysis (in particular, assessing the sufficiency of information in the software requirements specification). The unresolved nature of this problem necessitates the development of methods and information technology for assessing the sufficiency of information in the specifications of software requirements.

For assessing the sufficiency of the quality information in the software requirements specification, the specification should be analyzed to find the available attributes and/or quality indicators. There are many different methods of semantic analysis of software requirements specifications [127, 128] – Table 1.6.

Tab. 1.6. Known methods of semantic analysis of the software requirements [127]

Approach of semantic analysis of software requirements	Author(s)
Method of automatic assistance to developers through the transformation of the natural language requirements using UML diagrams of activity and sequence	S. Gulia et al. [129]
Method of the transformation of the natural language specifications into formal models, which are suitable for use during the development of information systems	M. Selway et al. [130]
The methodology, which consists of four processes: requirements' parsing, requirements' mapping using the matrix, the addition of requirements to specification template and third party inspection	S. W. Ali et al. [131]
The mechanism for automating the display of functional requirements into formal representations by means of marking a semantic role	T. Diaman-topoulos et al. [132]
Approach to automatically extracting the semantic information from the specifications of the software requirements by combining the methods of marking a semantic role and modeling of the domain knowledge	Y. Wang [133]
A technology, inputs of that are natural language artefacts (requirements) and that automatically detects relevant security sentences in artefacts and categorizes them according to security objectives	M. Riaz et al. [134]
Method for setting up and creating a combined analyzer for processing and analyzing the natural language specifications, that combines the benefits of formal analyzers, which are used to handle descriptions with a rigidly defined syntax (for example, source code), and analysts, which are designed to handle natural languages, that are well understand free text	F. Iwama et al. [135]
Methodology and tool QuARS Requirements Analysis Tool for systematic and automatic analysis of natural language requirements with the purpose of automatic detection of potential linguistic defects	S. Gnesi et al. [136]

An ontological approach to automated testing and measurement of software requirements, that is used to detect inconsistencies, inconsequences and deficiencies in software requirements	K. Siegemund [137]
The prototype of the semantic system, which is used to extract requirements using semi-formal representation	S. Farfeleder et al. [138]
An approach to maintaining the requirements engineering process in the software development process by using an ontology, which is developed to the features of the Scrum methodology	M. Murtazina et al. [139]
An approach based on a common model of users' requirements for integration of the heterogeneous requirements using ontologies	A. Mustafa et al. [140]
Web resources integration model involves the process decomposition on information integration, textual content syntax, semantics and structure for heterogeneous Web resources in intelligent information systems	J. Su et al. [141]

All the considered methods of semantic analysis of software requirements do not provide automated search of the attributes and/or indicators in the requirements, which are needed to determine the software characteristics and/or software metrics. The unresolved nature of this problem necessitates the development of a method of activity of an intelligent agent for semantic analysis of natural language requirements for the software.

1.5. Conclusions

The conducted study of the current state of information technology development has shown that at the current stage of development and implementation of information technology there are decisive changes caused by the need to process large amounts of information. The application of known methods and tools for processing such arrays of information leads to the loss of significant information. Today, the intellectualization of information and data processing, which allows solving new classes of problems on the basis of available information resources, is gaining significant development. Thus, there are prerequisites for the development of a methodology for creating and implementing new-generation information technologies. However, subject areas for which information technologies are developed significantly affect the methods of information processing, so an approach based on the study of characteristics and features of subject areas and the development of new information technologies specifically for specific industries is justified. Today, the software engineering industry needs special attention in the direction of

development and implementation of effective information technologies, in particular, in terms of software quality assurance.

A study of the impact of information in the requirements specification on software quality showed that the quality factors of modern software systems are less dependent on the writing of software code, but significantly depend on the formation of requirements. The risks of the insufficiently worked-out stage of requirements formation are non-compliance with project deadlines and financial overspending, which can lead to the closure of the project, or even the collapse of the software company due to its financial instability.

As shown above, it is at the end of the design phase that up to 55% of all future software errors can and should be detected and corrected. Defects introduced at the requirements' forming stage and design stage should be identified and remedied before they can affect the results of later stages of the life cycle. The quality and success of the software project significantly depend on the specification of software requirements.

Under such conditions, the analysis of the specification of software requirements, the ability to "cut off" software projects with incomplete (with insufficient information) specification is relevant and very important. Sufficiency of information in the requirements specifications is one of the most important aspects of software quality assessment in the early stages of the software life cycle.

When developing software projects, there is a gap in knowledge about the characteristics of future software. This gap appears due to the partial consideration of information of the subject area (including information of the specification of software requirements) when assessing the quality of software. The size of the knowledge gap is not constant for a software project. The probability that in the process of the project life cycle the knowledge gap will disappear is low, and with the appearance of new information of the subject area, there may be an increase in the size of the knowledge gap. For the safe operation of the software, it is desirable to reduce the size of the knowledge gap, taking into account as much the subject area information during the software life cycle. Therefore, fundamentally new approaches are needed, taking into account the subject area information when assessing the quality of software.

A study of known approaches to software quality assessment showed that:

1) the concepts used in the field of software quality assessment are subjectively dependent, and existing methods and tools for software quality assessment do not meet modern requirements for software systems;

2) there are no comprehensive methods that will provide an opportunity to assess not only the impact of each individual characteristic on the quality of software but also to assess the sufficiency of information to calculate quality characteristics, as well as to assess the interaction of characteristics in assessing software quality.

Today, two key approaches to software quality assessment are: the SQuaRE model (ISO 25010:2011 standard) and based on the results of the metric analysis.

Software quality assessment according to ISO 25010:2011 is as follows: on the basis of quality attributes specified in ISO 25023:2016, sub-characteristics and quality

characteristics are evaluated, which, in turn, provide a comprehensive assessment of software quality. One of the problems is the determination of the significance of quality attributes, which is influenced by the proven existence of the correlation of characteristics and sub-characteristics by attributes.

Evaluation of software quality and complexity based on the use of metric analysis results is as follows: based on indicators, the values of metrics are calculated, which, in turn, provide a comprehensive assessment of the quality and complexity of the software project and predicted assessment of the quality and complexity of future software. One of the problems of software quality assessment based on the results of the metric analysis is to determine the significance of indicators, which is influenced by the proven existence of correlation of metrics by indicators.

Therefore, the information on the quality of the software requirements specification consists of the quality attributes, as well as the quality and complexity indicators of the software, which are defined in the requirements specification. Then the sufficiency of information on quality in the specification of software requirements is the presence in the specification of all information elements (attributes and indicators), which are necessary to determine the quality.

Analysis of the ISO 25010 (SQuaRE) model and software quality and complexity metrics showed that a number of quality attributes are part of several sub-characteristics and characteristics, and a number of indicators are part of several metrics. Then, if such attributes and/or indicators are absent, their simultaneous use in determining the quality of the software will significantly affect the veracity of the obtained estimates of software quality, i.e. the correlation of characteristics by attributes and metrics by indicators may degrade the accuracy and veracity of software quality. In such a situation, it is important to mitigate the effect of cross-correlation of sub-characteristics and characteristics by attributes, as well as cross-correlation of metrics by indicators when using them in models or comprehensive software quality assessments.

Mitigation of the correlation is carried out by identifying joint attributes and/or indicators, ensuring their presence in the specification of software requirements, increasing the accuracy of their values, or, if possible, limiting the simultaneous use of characteristics and sub-characteristics that depend on the same attributes, and/or metrics containing the same indicators. The knowledge of experienced professionals on the interaction and correlation of sub-characteristics by attributes and metrics by indicators is valuable that should be stored and used when evaluating the specifications of software requirements for the sufficiency of quality information. Such knowledge should be presented in the form of ontologies that make it possible to reflect the causal relationships between concepts.

In this monograph, ontologies and weighted ontologies will be the theoretical basis for the process of assessing the sufficiency of information on quality in the software requirements specifications.

Analysis of known approaches, methods, information technologies, tools for software requirements analysis and for assessing the sufficiency of SRS information, and ontology-

based intelligent agents for the software engineering industry showed, that they do not solve the problem of assessing the sufficiency of quality information in the specification of software requirements. In addition, they all belong to different methodological approaches and do not integrate with each other, i.e. there is currently no intelligent information-analytical technology for improving the software quality by assessing the sufficiency of information in the early stages of the life cycle, which is one of the causes of problems in the software quality assurance domain. The actuality of the problem of analysis of the sufficiency of information on quality in software requirements specifications, as well as the lack of models, methods and tools for assessing the sufficiency of quality information in software requirements specifications, necessitates the development of intelligent information-analytical technologies for improving the software quality by assessing the sufficiency of information at the early stages of the life cycle.

Chapter 2. Intelligent Information Technology for Assessing the Sufficiency of Quality Information in the Specifications of the Software Requirements

2.1. Theoretical Basis for the Use of Ontologies to Assess the Sufficiency of Quality Information in Software Requirements Specifications

In Chapter 1, it was proved that quality information should be presented in the form of ontologies that allow identifying and displaying the causal links between the requirements governing the characteristics of quality, between directly the characteristics and sub-characteristics of software quality, quality metrics, etc.

For solving the problem of assessing the sufficiency of the amount of information on quality in the specifications of software requirements, the base (universal) ontologies of the subject area "Software Engineering" (parts "Software Quality", "Software Quality. Metric Analysis") were developed on the basis of ISO 25010:2011, ISO 25023:2016 (for quality assurance and evaluation according to the SQuaRE model) and industry publications (for quality assurance and evaluation based on metric information processing) [142-144]. Base ontologies reflect the necessary quality information (attributes and indicators) that must be available in the specification of software requirements to ensure the sufficiency of its quality information.

In the process of developing concrete software, in addition to base ontologies, it is necessary to have an ontology of this software, which reflects the available information on quality (attributes and indicators) in the specification of requirements for concrete software.

Comparison of the developed ontology of concrete software with base ontologies allows determining the insufficiency (Fig. 2.1) or sufficiency (Fig. 2.2) of quality information in the specification of requirements for concrete software.

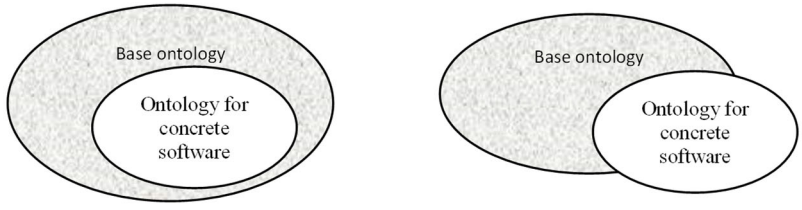

Fig. 2.1. Two cases of insufficiency of requirements' information

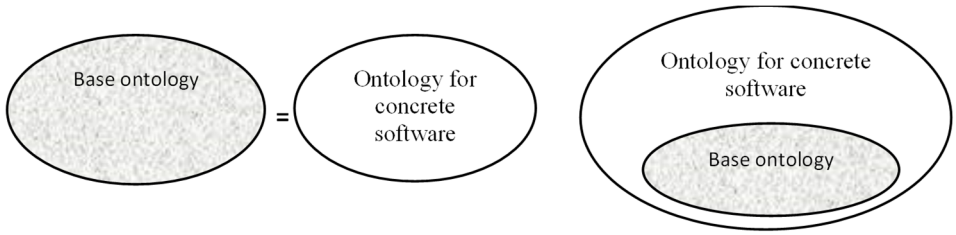

Fig. 2.2. Two cases of sufficiency of requirements' information

The criterion of the sufficiency of quality information in the software requirements specifications. Let $SAMI$ is the set of missing attributes and indicators, and

$$SAMI = Base\ ontology \setminus (Base\ ontology \cap Ontology\ of\ specific\ software) . \quad (2.1)$$

Then:

1) if $SAMI = \varnothing$, then the quality information in the software requirements specification is sufficient;

2) if $SAMI \neq \varnothing$, then the quality information in the software requirements specification is insufficient, and the requirements specification needs to be supplemented with quality information (attributes and (or) indicators).

For determining the structure and content of base (universal) ontologies of the subject area "Software Engineering" (parts "Software Quality", "Software Quality. Metric Analysis"), ontological software quality models are required (according to ISO 25010 and using the results of metric analysis). In addition, taking into account the presented criterion of the sufficiency of quality information in the software requirements specifications, it is necessary to develop models of the process of assessing the sufficiency of information for determining the quality of software based on ISO 25010:2011 and using the results of the metric analysis.

Ontological models of software quality (according to ISO 25010). Formally, the ontology is defined as: $O =< X, RX, F >$, where X – a finite set of concepts of the subject area, RX – a finite set of relations between concepts, F – a finite set of interpretation functions given on concepts or relations.

We present a model of the subject area "Software Engineering" (part "Software Quality") based on the ontology in the form of: $O_Q =< X_Q, RX_Q, F_Q >$, where X_Q – a finite set of characteristics, sub-characteristics and attributes of software quality, RX_Q – a finite set of relations between concepts, F_Q – a finite set of interpretation functions defined for the characteristics, sub-characteristics and quality attributes of the software.

The set of characteristics, sub-characteristics and attributes of software quality:

$$X_Q = \{ QCH, QSCH, QMS \} = \{ x_Q^1, ..., x_Q^{177} \}, \quad (2.2)$$

where: $\{x_Q^1,...,x_Q^8\} \in QCH$ – the set of software quality characteristics, i.e. $\{x_Q^1,...,x_Q^8\} = \{qch_1,...qch_8\}$;

$\{x_Q^9,...,x_Q^{39}\} \in QSCH$ – the set of software quality sub-characteristics, i.e. $\{x_Q^9,...,x_Q^{39}\} = \{qsch_1,...,qsch_{31}\}$;

$\{x_Q^{40},...,x_Q^{177}\} \in QMS$ – the set of software quality attributes, i.e. $\{x_Q^{40},...,x_Q^{177}\} = \{qms_1,...,qms_{138}\}$.

The set of relations between concepts RX_Q consists of the relation "depends on", i.e. $RX_Q = \{"depends\ on"\}$.

The set F_Q of interpretation functions, which are defined for the characteristics, sub-characteristics and quality attributes of the software, consists of functions of the dependence of quality on characteristics, characteristics on sub-characteristics, sub-characteristics on quality attributes, i.e. $F_Q = \{f_Q^1,...,f_Q^{40}\} = \{f(), f_1(),..., f_8(), \varphi_1(),..., \varphi_{31}()\}$

Then *the base (universal) model of the subject area "Software Engineering" (part "Software Quality") based on the ontology*, taking into account the formula (2.2), has the form:

$$O_Q = \begin{Bmatrix} qch_1,...,qch_8, qsch_1,...,qsch_{31}, qms_1,...,qms_{138}, \\ "depends\ on", f(), f_1(),..., f_8(), \varphi_1(),..., \varphi_{31}() \end{Bmatrix}. \quad (2.3)$$

The ontology (ontological knowledge base) for the model (2.3) is filled on the basis of information taken from the standards [31, 32].

Since when determining the quality of the software it is necessary to take into account the weights of quality attributes, there is a need to include such weights in the base (universal) model of the subject area "Software Engineering" (part "Software Quality") and to develop the weighted ontology.

Then the set of characteristics, sub-characteristics and attributes of software quality for the software quality model according to ISO 25010:2011 based on a weighted ontology will be as follows:

$$X_{Qw} = \{x_{Qw}^1,...,x_{Qw}^{177}\}, \quad (2.4)$$

where: $\{x_{Qw}^1,...,x_{Qw}^8\} \in QCH$, і $\{x_{Qw}^1,...,x_{Qw}^8\} = \{qch_1,...qch_8\}$;

$\{x_{Qw}^9,...,x_{Qw}^{39}\} \in QSCH$, і $\{x_{Qw}^9,...,x_{Qw}^{39}\} = \{qsch_1,...,qsch_{31}\}$;

$\{x_{Qw}^{40},...,x_{Qw}^{177}\} = \{(qms_1, w_1^m),...,(qms_{138}, w_{138}^m)\}$, and the subset $\{x_{Qw}^{40},...,x_{Qw}^{177}\}$ consists of pairs (qms_i, w_i^m), where qms_i – i-th software quality attribute ($qms_i \in QMS$), w_i^m – weight of i-th software quality attribute, $i = \overline{1,\ 138}$.

The base (universal) model of the subject area "Software Engineering" (part "Software Quality") based on a weighted ontology, taking into account the formula (2.4), has the following form:

$$O_Q^w = \begin{Bmatrix} qch_1,...,qch_8, qsch_1,...,qsch_{31}, (qms_1, w_1^m),...,(qms_{138}, w_{138}^m), \\ "depends\ \ on", f(), f_1(),..., f_8(), \varphi_1(),..., \varphi_{31}() \end{Bmatrix}. \quad (2.5)$$

The weighted ontology (ontological knowledge base), which is developed according to model (2.5), is filled on the basis of information taken from the standards [31, 32], taking into account the weights of software quality attributes, which were calculated in Chapter 1, taking into account the relationships between characteristics and sub-characteristics of software quality by attributes.

The ontological model of quality of concrete software has the form:

$$O_Q^{real} = \begin{Bmatrix} qch_1,...,qch_{nch}, qsch_1,...,qsch_{nsch}, qms_1,...,qms_{nm}, \\ "depends\ \ on", f(), f_1(),..., f_8(), \varphi_1(),..., \varphi_{31}() \end{Bmatrix}, \quad (2.6)$$

where nch ($nch \le 8$) – the number of software quality characteristics that can be calculated based on the available quality attributes in the specification of the requirements for concrete software, $nsch$ ($nsch \le 31$) – the number of software quality sub-characteristics that can be calculated based on the available quality attributes in the specification of the requirements for concrete software, nm ($nm \le 138$) – the number of quality attributes available in the specification of the requirements for the concrete software.

The ontology (ontological knowledge base), which is developed according to the model (2.6), is filled on the basis of information taken from the specification of requirements for concrete software.

Model of the process of assessing the sufficiency of information for determining the software quality according to ISO 25010. Given the criterion for assessing the sufficiency of information on quality in the specifications of software requirements, base (universal) models of the subject area "Software Engineering" (part "Software Quality") based on the ontology and weighted ontology, represented by formulas (2.3), (2.5), as well as the ontological model of quality of concrete software, represented by formula (2.6), we develop the model of the process of assessing the sufficiency of information for determining the software quality based on ISO 25010.

The process of assessing the sufficiency of information for determining the software quality consists of:

1) comparison of ontology for determining the quality of concrete software with the base (universal) ontology in order to identify attributes, which are missing in the ontology for determining the quality of concrete software, i.e. missing in the specification of software

requirements, for which this ontology was developed, as well as identifying characteristics and sub-characteristics of software quality that cannot be calculated on the basis of the available attributes in the specification of requirements for concrete software;

2) forming a conclusion about the insufficiency of quality information in the specification of software requirements, if in the specification of software requirements some attributes are missing, and there are characteristics and sub-characteristics of software quality that can not be calculated based on available attributes in the requirements;

3) comparison of ontology for determining the quality of concrete software with a weighted base (universal) ontology, if a conclusion about the insufficiency of information was formed – with the purpose of establishing the weights of attributes, which are missing in the specification of requirements for concrete software, to further prioritize the addition of attributes to the specification of the software requirements.

The scheme of the process of assessing the sufficiency of information for determining the software quality according to ISO 25010:2011 is presented in Fig. 2.3.

Model of the process of assessing the sufficiency of information for determining the software quality according to ISO 25010. Let $SMIQ = O_Q \setminus (O_Q \cap O_Q^{real})$, where: $SMIQ = \{qch_1,...,qch_{(8-nch)}, qsch_1,...,qsch_{(31-nsch)}, qms_1,...,qms_{(138-nm)}\}$, $\{qch_1,...,qch_{(8-nch)}\}$ – a subset of software quality characteristics that cannot be calculated based on the available quality attributes in the specification of requirements for the concrete software; $\{qsch_1,...,qsch_{(31-nsch)}\}$ – a subset of software quality sub-characteristics that cannot be calculated based on the available quality attributes in the specification of requirements for the concrete software; $\{qms_1,...,qms_{(138-nm)}\}$ – a subset of attributes that are not in the specification of the requirements for the concrete software.

Taking into account the criterion of the sufficiency of information on quality in the specifications of software requirements, we develop a *production rule for deciding on the sufficiency or insufficiency of information on quality (attributes)*:

$$\begin{aligned} &if\ SMIQ = \varnothing\ then\ "SRS\ information\ is\ sufficient" \\ &else: "SRS\ information\ is\ insufficient", SMMW = O_Q^w \setminus (O_Q^w \cap O_Q^{real}) \end{aligned}, \quad (2.7)$$

where $SMMW \supset \{(qms_1, w_1^m),...,(qms_{(138-nm)}, w_{(138-nm)}^m)\}$ – a subset of the attributes, which are not specified in the specification of the requirements for the concrete software, and their weights.

Numerical assessment of the sufficiency of the amount of quality information (attributes), which is available in the specification of requirements, can be calculated by the formula:

$$D_{chr} = \frac{ksc - \sum_{i=1}^{ksc} \frac{qtmm_i}{qtnm_i}}{ksc}, \quad (2.8)$$

where D_{chr} – sufficiency of the information (attributes), which is available in the specification, for software quality assessment according to the ISO 25010 standard, $qtmm_i$ – the number of attributes missing in the requirements for the i-th software quality characteristic, $qtnm_i$ – the number of required attributes for the i-th software quality characteristic, $\sum_{i=1}^{ksc} qtnm_i = 203$ – the total number of attributes (including repetitive ones), on which the software quality characteristics depend, ksc – number of software quality characteristics (according to ISO 25010:2011, $ksc = 8$).

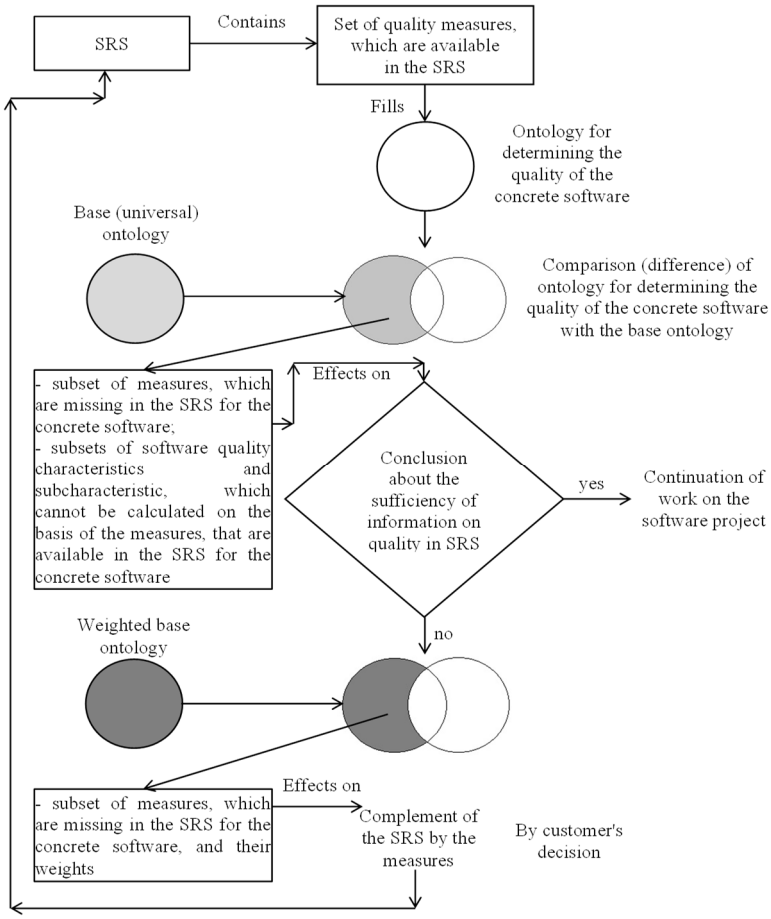

Fig. 2.3. The scheme of the process of assessing the sufficiency of information for determining the software quality according to ISO 25010:2011[75, 76]

The developed model of the process of assessing the sufficiency of information for determining the software quality according to ISO 25010:2011 reflects the features of assessing the sufficiency of information for determining the software quality according to ISO 25010:2011, provides the adaptation to specifics of the subject area and provides the possibility of access, analysis and understanding of information not only by man, but also virtual agents (bots), and are also a theoretical basis for the development of methods and tools for assessing the sufficiency of information on quality in the specifications of software requirements.

Ontological models of software quality (based on metric analysis). The model of the subject area "Software Engineering" (part "Software Quality. Metric Analysis") based on the ontology has the form: $O_{metr} =< X_{metr}, RX_{metr}, F_{metr} >$, where X_{metr} – a finite set of metrics and indicators of software quality and complexity, RX_{metr} – a finite set of relationships between concepts, F_{metr} – a finite set of interpretation functions defined for metrics and indicators of software quality and complexity.

The set of metrics and indicators of software quality and complexity:

$$X_{metr} = \{ SQM, SCXM, SQCXI \} = \{ x^1_{metr}, ..., x^{66}_{metr} \}, \quad (2.9)$$

where: $\{ x^1_{metr}, ..., x^{14}_{metr} \} \in SQM$ – a set of software quality metrics at the design stage (and $\{ x^1_{metr}, ..., x^5_{metr} \} \in SQM_{exv}$, $\{ x^6_{metr}, ..., x^{14}_{metr} \} \in SQM_{prv}$), $\{ x^{15}_{metr}, ..., x^{24}_{metr} \} \in SCXM$ – a set of software complexity metrics at the design stage (and $\{ x^{15}_{metr}, ..., x^{18}_{metr} \} \in SCXM_{exv}$, $\{ x^{19}_{metr}, ..., x^{24}_{metr} \} \in SCXM_{prv}$), $\{ x^{25}_{metr}, ..., x^{66}_{metr} \} \in SQCXI$ – a set of software quality indicators.

The set of relationships between concepts RX_{metr} consists of the relation "depends on", i.e. $RX_{metr} = \{ "depends\ on" \}$.

The set F_{metr} of interpretation functions, which are defined for software quality, consists of functions of quality dependence on quality metrics, metrics on quality indicators, i.e. $F_{metr} = \{ f^1_{metr}, ..., f^{26}_{metr} \} = \{ \psi(), \psi_1(), ..., \psi_{14}(), \phi(), \phi_1(), ..., \phi_{10}() \}$.

Then, taking into account the formula (2.9), *the base (universal) model of the subject area "Software Engineering" (part "Software Quality. Metric Analysis") based on the ontology* has the form:

$$O_{metr} = \begin{Bmatrix} sqm_1, ..., sqm_{14}, scxm_1, ..., scxm_{10}, sqcxi_1, ..., sqcxi_{42}, \\ "depends\ on", \psi(), \psi_1(), ..., \psi_{14}(), \phi(), \phi_1(), ..., \phi_{10}() \end{Bmatrix}. \quad (2.10)$$

Since, when determining the quality of software based on metric information, it is necessary to take into account the weights of quality indicators, there is a need to include such weights in the base (universal) model for determining the quality and complexity of software using metric analysis and to develop the weighted ontology.

Then the set of metrics and indicators of quality and complexity of the software of the model for determining the software quality using the results of metric analysis based on a weighted ontology will be as follows:

$$X_{metrw} = \{ x^{1}_{metrw},...,x^{38}_{metrw} \}, \quad (2.11)$$

where: $\{ x^{1}_{metrw},...,x^{14}_{metrw} \} \in SQM$, $\{ x^{15}_{metrw},...,x^{24}_{metrw} \} \in SCXM$, $\{ x^{25}_{metr},...,x^{66}_{metr} \} = \{ (sqcxi_1, w^{ind}_1),...,(sqcxi_{42}, w^{ind}_{42}) \}$, that is, a subset $\{ x^{25}_{metr},...,x^{66}_{metr} \}$ consists of pairs $(sqcxi_i, w^{ind}_i)$, where $sqcxi_i$ – i-th indicator of software quality ($sqcxi_i \in SQCXI$), w^{ind}_i – weight of i-th indicator, $i = \overline{1,\ 42}$.

The base (universal) model of the subject area "Software Engineering" (part "Software Quality. Metric Analysis") based on a weighted ontology, taking into account the formula (2.11), has the following form:

$$O^{w}_{metr} = \begin{Bmatrix} sqm_1,...,sqm_{14}, scxm_1,...,scxm_{10}, (sqcxi_1, w^{ind}_1),...,(sqcxi_{42}, w^{ind}_{42}), \\ "depends\ on", \psi(), \psi_1(),..., \psi_{14}(), \phi(), \phi_1(),..., \phi_{10}() \end{Bmatrix}. \quad (2.12)$$

The weighted ontology (ontological knowledge base), which is developed according to the model (2.12), is filled taking into account the weights of software quality indicators, which were calculated in Chapter 1, taking into account the relationship between software quality and complexity metrics by the indicators.

The ontological model for determining the quality of concrete software (based on metric information) has the form:

$$O^{real}_{metr} = \begin{Bmatrix} sqm_1,...,sqm_{nqm}, scxm_1,...,scxm_{ncxm}, sqcxi_1,...,sqcxi_{nqcxi}, \\ "depends\ on", \psi(), \psi_1(),..., \psi_{14}(), \phi(), \phi_1(),..., \phi_{10}() \end{Bmatrix}, \quad (2.13)$$

where nqm ($nqm \le 14$) – the number of software quality metrics that can be calculated based on the indicators, which are available in the specification of requirements for the concrete software, $ncxm$ ($ncxm \le 10$) – the number of software complexity metrics that can be calculated based on the indicators, which are available in the specification of requirements for the concrete software, $nqcxi$ ($nqcxi \le 42$) – the number of quality indicators, which are available in the specification of software requirements.

The ontology (ontological knowledge base), which is developed according to the model (2.13), is filled on the basis of information taken from the specification of requirements for the concrete software.

Model of the process of assessing the sufficiency of information for determining the software quality using the results of the metric analysis. Taking into account the criterion of assessing the sufficiency of quality information in the software requirements specifications, as well as developed base (universal) models of the subject area "Software Engineering" (part "Software Quality. Metric Analysis") based on ontology and weighted ontology, presented by formulas (2.10), (2.12), as well as an ontological model for determining the quality of the concrete software using the results of the metric analysis,

presented by formula (2.13), we develop a model of the process of assessing the sufficiency of information for determining the software quality using the results of the metric analysis.

The process of assessing the sufficiency of information for determining the software quality using the results of the metric analysis consists of:

1) comparison of ontology for determining the quality of concrete software using the results of the metric analysis with the base (universal) ontology in order to identify indicators, which are missing in the ontology for determining the quality of concrete software (based on metric information), i.e. missing in the specification of software requirements, for which this ontology was developed, as well as identifying metrics of software quality and complexity that cannot be calculated on the basis of the available indicators in the specification of requirements for concrete software;

2) forming a conclusion about the insufficiency of quality information in the specification of software requirements, if in the specification of software requirements some indicators are missing, and there are metrics of software quality and complexity that can not be calculated based on available indicators in the requirements;

3) comparison of ontology for determining the quality of concrete software with a weighted base (universal) ontology, if a conclusion about the insufficiency of information was formed – with the purpose of establishing the weights of indicators, which are missing in the specification of requirements for concrete software, to further prioritize the addition of indicators to the specification of the software requirements.

Model of the process of assessing the sufficiency of information for determining the software quality using the results of the metric analysis. Let $SMIQM = O_{metr} \setminus (O_{metr} \cap O_{metr}^{real})$, where:

$SMIQM = \{ sqm_1,...,sqm_{(14-nqm)}, scxm_1,...,scxm_{(10-ncxm)}, sqcxi_1,...,sqcxi_{(42-nqcxi)} \}$

$\{ sqm_1,...,sqm_{(14-nqm)} \}$ – a subset of software quality metrics that cannot be calculated based on available indicators in the specification of software requirements; $\{ scxm_1,...,scxm_{(10-ncxm)} \}$ – a subset of software complexity metrics that cannot be calculated based on available indicators in the specification of software requirements; $\{ sqcxi_1,...,sqcxi_{(42-nqcxi)} \}$ – a subset of indicators that are missing in the specification of requirements for concrete software.

Taking into account the criterion of the sufficiency of information on quality in the specifications of software requirements, we will develop *a production rule for deciding on the sufficiency or insufficiency of information on quality (indicators)*:

$$\begin{array}{l} if\ SMIQM = \varnothing\ then\ "SRS\ information\ is\ sufficient" \\ else: "SRS\ information\ is\ insufficient", SMIW = O_{metr}^{w} \setminus (O_{metr}^{w} \cap O_{metr}^{real}) \end{array}, \quad (2.14)$$

where $SMIW \supset \{(sqcxi_1, w_1^{ind}),...,(sqcxi_{(42-nqcxi)}, w_{(42-nqcxi)}^{ind})\}$ – a subset of the indicators, missing in the specification of the requirements for the concrete software, and their weights.

Numerical assessment of the sufficiency of the amount of quality information (indicators) available in the specification of requirements can be calculated by the formula:

$$D_{metr} = \frac{ksm - \sum_{i=1}^{ksm} \frac{qtmmetr_i}{qtnmetr_i}}{ksm}, \quad (2.15)$$

where D_{metr} – sufficiency of the volume of information (indicators) available in the specification for evaluation of software quality using the results of metric analysis, $qtmmetr_i$ – the number of missing indicators in the specification for the i-th software metric, $qtnmetr_i$ – the number of required indicators for the i- th software metric, $\sum_{i=1}^{ksm} qtnmetr_i = 72$ – the total number of indicators (including repetitive ones) on which the software metrics depend, ksm – the number of metrics with exact and predicted values at the design stage of the software (metric analysis showed that $ksm = 24$).

The developed model of the process of assessing the sufficiency of information for determining the software quality using the results of the metric analysis is designed to support the process of assessing the quality of software in the early stages of the life cycle, provides a conclusion on the sufficiency of information, as well as is a theoretical basis for the development of methods and tools for assessing the sufficiency of information on quality in the specifications of software requirements.

Ontological model of software requirements specification (based on ISO 29148:2018 [145]). The ontological model of the subject area "Software Engineering" (part "Specification of software requirements (quality attributes)") has the form: $O_{SRS} =< X_{SRS}, RX_{SRS} >$, where X_{SRS} – a finite set of concepts (specification sections and software quality attributes), RX_{SRS} – a finite set of relationships between concepts.

The set of concepts:

$$X_{SRS} = \{ SRS, QMS \} = \{ x_{SRS}^1, ..., x_{SRS}^{143} \}, \quad (2.16)$$

where $\{ x_{SRS}^1, ..., x_{SRS}^5 \} = \{ R1, ..., R5 \}$ – a set of sections of the specification, $\{ x_{SRS}^6, ..., x_{SRS}^{143} \} = \{ qms_1, ..., qms_{138} \}$ – a set of quality attributes in the specification.

The set of relationships between concepts RX_{SRS} consists of the relation "contained in", i.e. $RX_{SRS} = \{ "contained\ \ in" \}$.

Then, taking into account the formula (2.16), *the base (universal) ontological model of the subject area "Software Engineering" (part "Specification of software requirements (quality attributes)")* has the form:

$$O_{SRS} = \{ R1, ..., R5, qms_1, ..., qms_{138}, "contained\ \ in" \}. \quad (2.17)$$

The ontological model of the subject area "Software Engineering" (part "Specification of software requirements (quality indicators)") has the form:

$O_{SRSmetr} =< X_{SRSmetr}, RX_{SRSmetr} >$, where $X_{SRSmetr}$ – a finite set of concepts (specification sections and software quality indicators), $RX_{SRSmetr}$ – a finite set of relationships between concepts.

The set of concepts:

$$X_{SRSmetr} = \{ SRS_{metr}, SQCXI \} = \{ x^{1}_{SRSmetr}, ..., x^{47}_{SRSmetr} \}, \quad (2.18)$$

where $\{ x^{1}_{SRSmetr}, ..., x^{5}_{SRSmetr} \} = \{ R1_{metr}, ..., R5_{metr} \}$ – a set of sections of the specification, $\{ x^{6}_{SRSmetr}, ..., x^{47}_{SRSmetr} \} = \{ sqcxi_1, ..., sqcxi_{42} \}$ – a set of quality indicators in the specification.

The set of relationships between concepts $RX_{SRSmetr}$ consists of the relation "contained in", i.e. $RX_{SRSmetr} = \{ "contained\ \ in" \}$.

Then, taking into account the formula (2.18), *the base (universal) ontological model of the subject area "Software Engineering" (part "Specification of software requirements (quality indicators)")* has the form:

$$O_{SRSmetr} = \{ R1_{metr}, ..., R5_{metr}, sqcxi_1, ..., sqcxi_{42}, "contained\ \ in" \}. \quad (2.19)$$

The developed ontological models of software requirements specification (in terms of availability of quality information) are in fact templates of specification of requirements in terms of availability in it the quality information, which can be processed by both specialists and automated tools or intelligent agents (bots).

2.2. Methods for Assessing the Sufficiency of Quality Information (According to ISO 25010:2011) in Software Requirements Specifications

Method for assessing the sufficiency of quality information (according to ISO 25010:2011) in software requirements specifications based on ontology [74-76, 142-144]. For assessing the sufficiency of quality information (according to ISO 25010:2011) in the software requirements specifications, the method based on comparative analysis of the ontologies is proposed. First of all, for the subject area "Software Engineering" we develop a base ontology, which contains information on the characteristics, sub-characteristics and attributes of software quality.

The concept of software quality assessment is presented in Fig. 2.4. The concept of the base ontology of the subject area "Software Engineering" (part "Software Quality") is presented in Fig. 2.5. The base ontology of the subject area "Software Engineering" (part "Software Quality") is formed by components for Functional Suitability (Fig. 2.6), Compatibility (Fig. 2.7), Performance Efficiency (Fig. 2.8), Portability (Fig. 2.9), Usability (Fig. 2.10), Reliability (Fig. 2.11), Security (Fig. 2.12), Maintainability (Fig. 2.13).

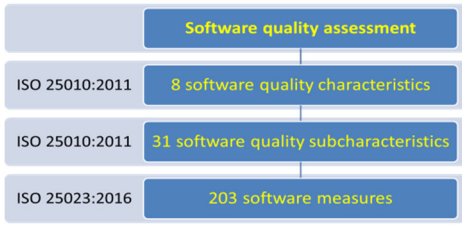

Fig. 2.4. The modern concept of software quality assessment by ISO 25010:2011

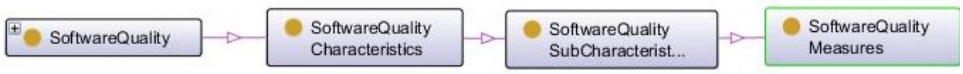

Fig. 2.5. Concept of the base ontology of the subject area "Software Engineering" (part "Software Quality")

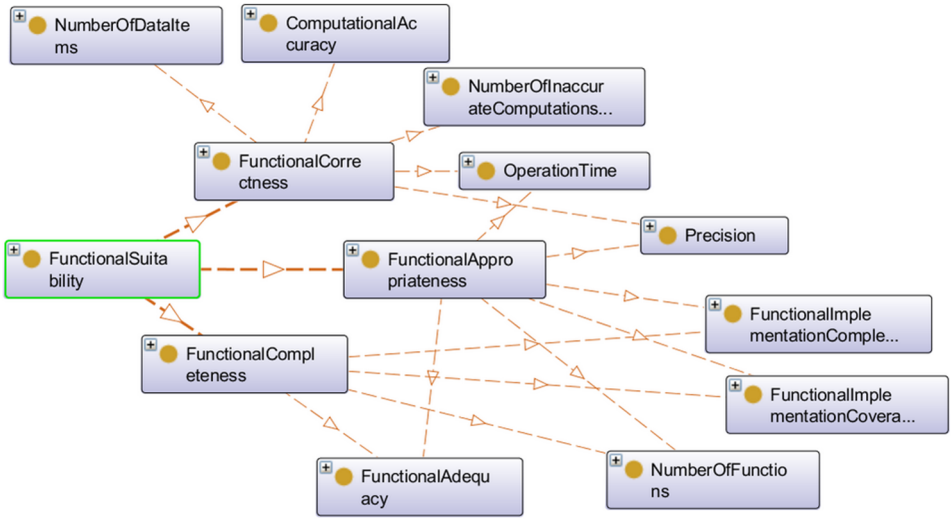

Fig. 2.6. Component of the base ontology for Functional Suitability

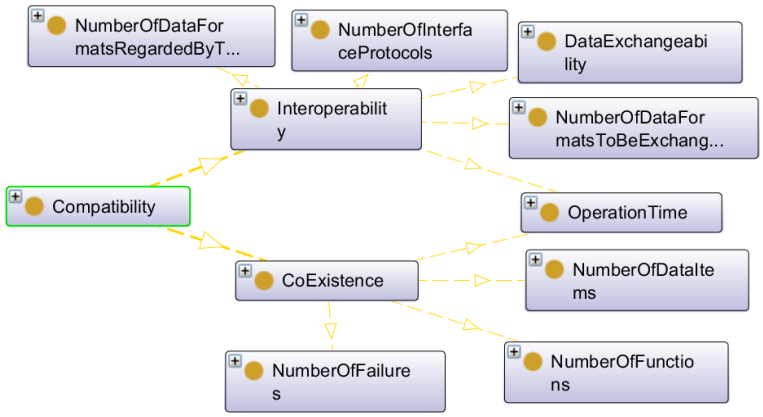

Fig. 2.7. Component of the base ontology for Compatibility

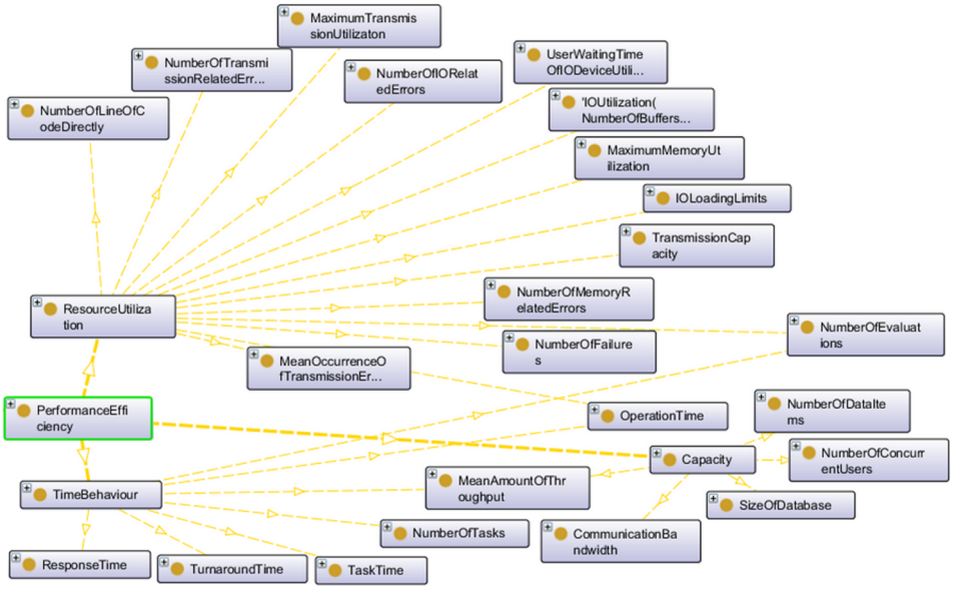

Fig. 2.8. Component of the base ontology for Performance Efficiency

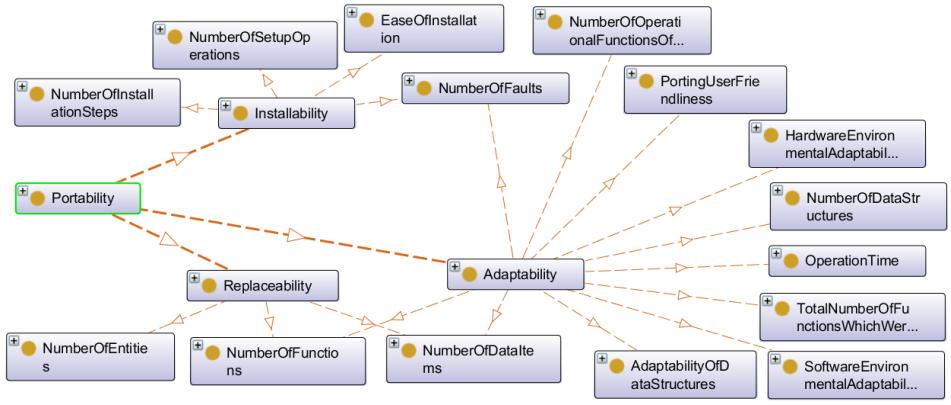

Fig. 2.9. Component of the base ontology for Portability

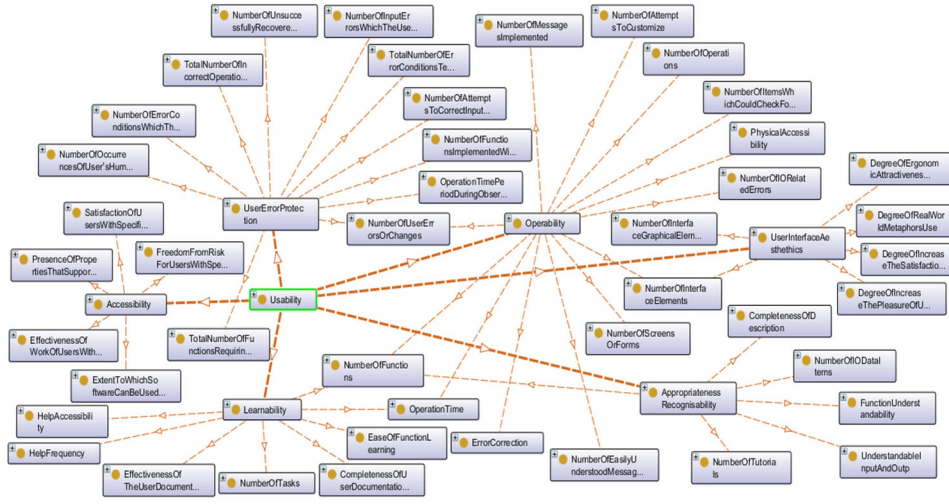

Fig. 2.10. Component of the base ontology for Usability

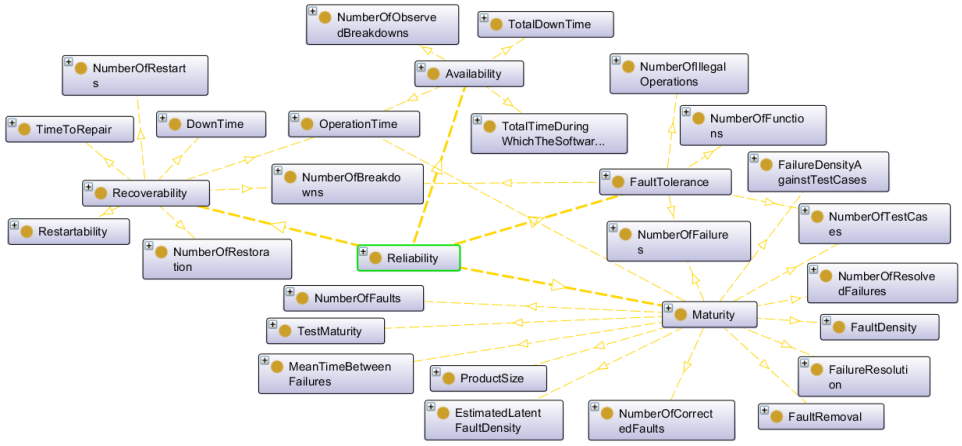

Fig. 2.11. Component of the base ontology for Reliability

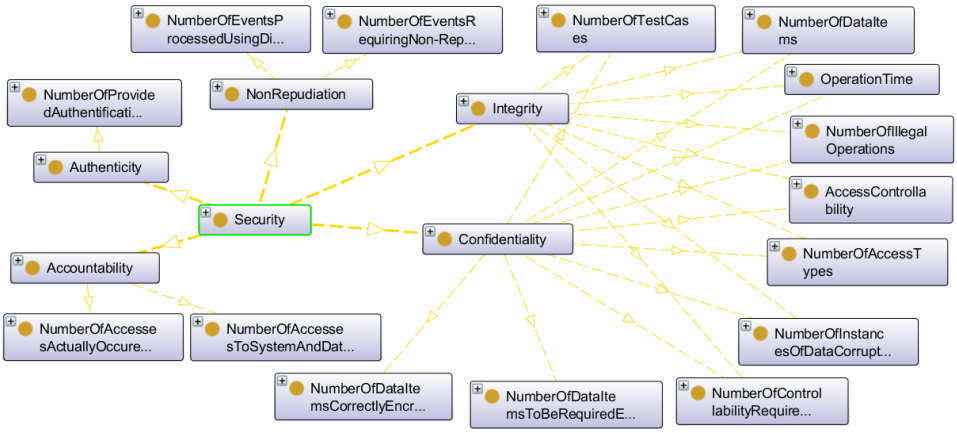

Fig. 2.12. Component of the base ontology for Security

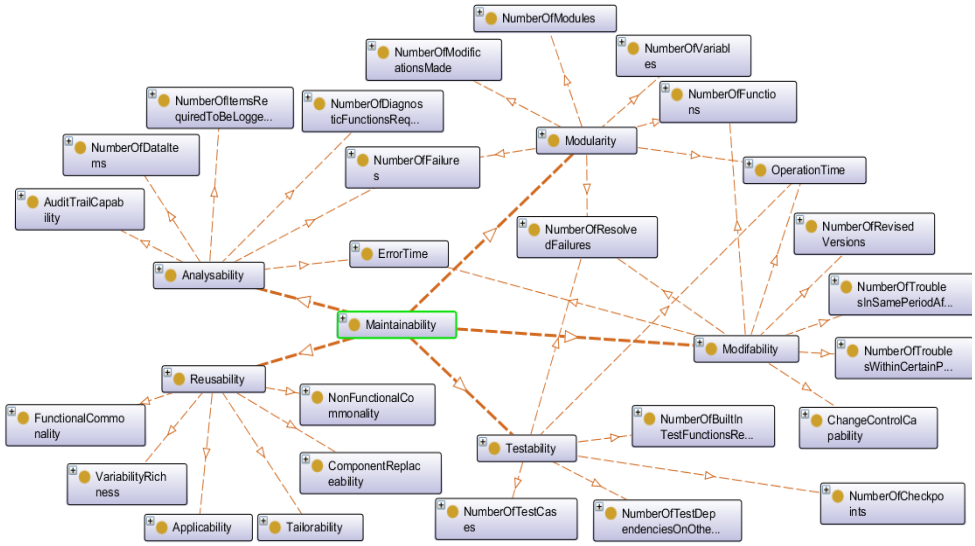

Fig. 2.13. Component of the base ontology for Maintainability

Method for assessing the sufficiency of quality information (according to ISO 25010:2011) in software requirements specifications based on ontology consists of the following stages:

1) analysis of sections of the specification of requirements for the concrete software for the presence of attributes necessary to determine the software quality sub-characteristics and characteristics, i.e. for the presence of elements of the subset $\{qms_1,...,qms_{nm}\} \in O_Q^{real}$;

2) generating and filling the template of ontology for determining the quality of concrete software, which is formed by components for Functional Suitability, Performance Efficiency, Usability, Reliability, Compatibility, Security, Maintainability, Portability of the concrete software, i.e. generating and filling the template of ontology $O_Q^{real} = \begin{Bmatrix} qch_1,...,qch_{nch}, qsch_1,...,qsch_{nsch}, qms_1,...,qms_{nm}, \\ "depends\ on", f(), f_1(),...,f_8(), \varphi_1(),...,\varphi_{31}() \end{Bmatrix}$;

3) comparison of the obtained ontology with the base ontology of the subject area "Software Engineering" (part "Software Quality"), the components of which are shown in Fig. 2.6-2.13, i.e. comparison of the set of attributes $\{qms_1,...,qms_{nm}\}$ from the ontology for determining the quality of concrete software $O_Q^{real} = \begin{Bmatrix} qch_1,...,qch_{nch}, qsch_1,...,qsch_{nsch}, qms_1,...,qms_{nm}, \\ "depends\ on", f(), f_1(),...,f_8(), \varphi_1(),...,\varphi_{31}() \end{Bmatrix}$ with the corresponding set

$\{qms_1,...,qms_{138}\}$ from the base ontology of the subject area "Software Engineering" (part "Software Quality") $O_Q = \begin{Bmatrix} qch_1,...,qch_8, qsch_1,...,qsch_{31}, qms_1,...,qms_{138}, \\ "depends\ on", f(), f_1(),...,f_8(), \varphi_1(),...,\varphi_{31}() \end{Bmatrix}$;

4) identifying attributes that are missing in the ontology for determining the quality of concrete software, i.e. the formation of a subset $\{qms_1,...,qms_{(138-nm)}\} = \{qms_1,...,qms_{138}\} \setminus (\{qms_1,...,qms_{138}\} \cap \{qms_1,...,qms_{nm}\})$, where $\{qms_1,...,qms_{(138-nm)}\} \in SMIQ$, $\{qms_1,...,qms_{138}\} \in O_Q$, $\{qms_1,...,qms_{nm}\} \in O_Q^{real}$ (if the generated set is not empty, then the information in the specification is not sufficient for calculating the quality sub-characteristics and characteristics – the more elements in the specified set, the lower the level of sufficiency of information in the requirements);

5) identification (on the basis of comparative analysis of the developed ontologies) of quality sub-characteristics and characteristics that cannot be calculated on the basis of available attributes, i.e. formation of subsets $\{qch_1,...,qch_{(8-nch)}\} \in SMIQ$, $\{qsch_1,...,qsch_{(31-nsch)}\} \in SMIQ$ (herewith it should be borne in mind the main idea of ISO 25010, which is that quality assessment should be carried out comprehensively, taking into account all these quality characteristics, assessment of quality characteristics should also be carried out comprehensively, taking into account all these sub-characteristics, assessment of quality sub-characteristics, in turn, should be carried out comprehensively, taking into account all these attributes);

6) the presence of sub-characteristics and characteristics, the value of which cannot be determined on the basis of attributes, which are available in the specification of software requirements, indicates the insufficiency of information for reliable assessment of software quality, i.e. the need to supplement this specification with attributes, which are needed for calculating the one or another characteristic;

7) forming a request to the developers of specifications for the requirements governing the attributes of quality; making the necessary additions to the specification with quality information (attributes);

8) repeating steps 1-7 until all attributes are included in the requirements specification and until it is possible to define all sub-characteristics and quality characteristics, or until the conclusion about insufficiency of information on quality in the software requirements specification satisfies the customer.

The final conclusion about the insufficiency of quality information in the specification of software requirements will be taken if the cost of finalizing the specification of software requirements becomes greater than the expected effect on quality assessment.

We present the developed method for assessing the sufficiency of quality information (according to ISO 25010:2011) in software requirements specifications based on ontology in the form of a scheme – Fig. 2.14 [144].

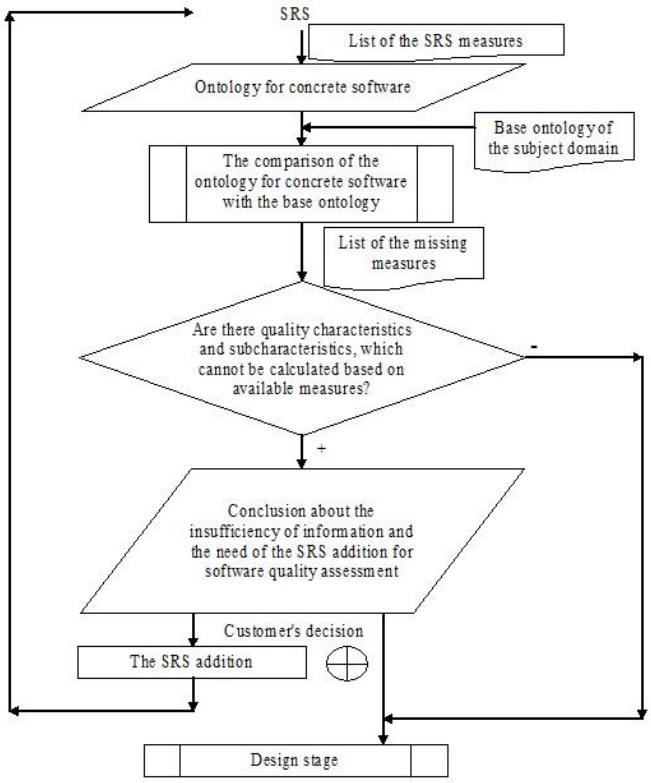

Fig. 2.14. Scheme of the method for assessing the sufficiency of quality information (according to ISO 25010:2011) in software requirements specifications based on ontology [144]

Method for assessing the sufficiency of quality information (according to ISO 25010:2011) in software requirements specifications based on weighted ontology [74-76, 142-144]. When assessing the quality of software according to ISO 25010:2011 for ensuring the appropriate level of veracity, it is important to meet the presence in the specification of software requirements for those attributes that have higher weights.

Definition 2.1. A weighted ontology of the subject area "Software Engineering" in the part "Software Quality" will be called an ontology, in which software quality attributes have weights in order to recommend further satisfaction of these attributes in the specification of software requirements.

Based on the base ontology of the subject area "Software Engineering" (part "Software Quality"), the components of which are presented in Fig. 2.6- 2.13, we construct the weighted base ontology, which contains information on the weights of software quality attributes (Fig. 2.15-2.22).

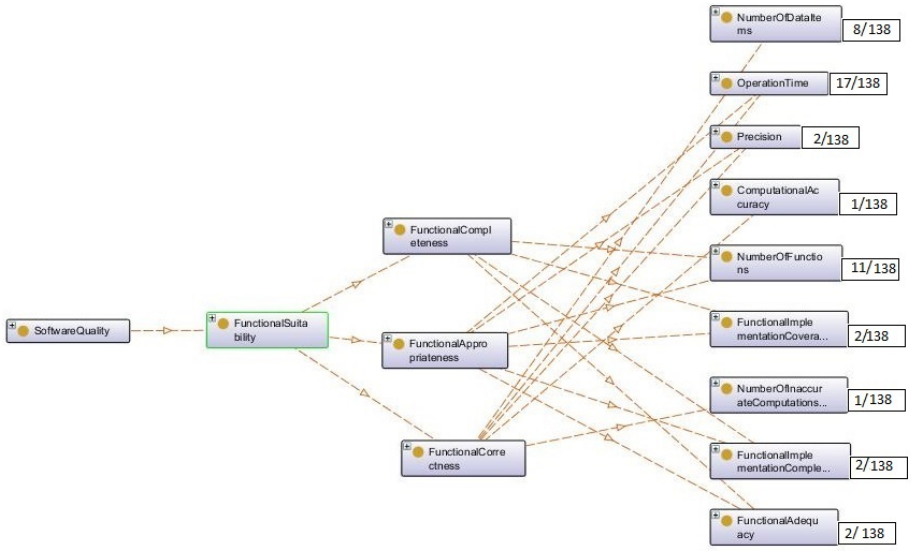

Fig. 2.15. Component of the weighted base ontology for Functional Suitability

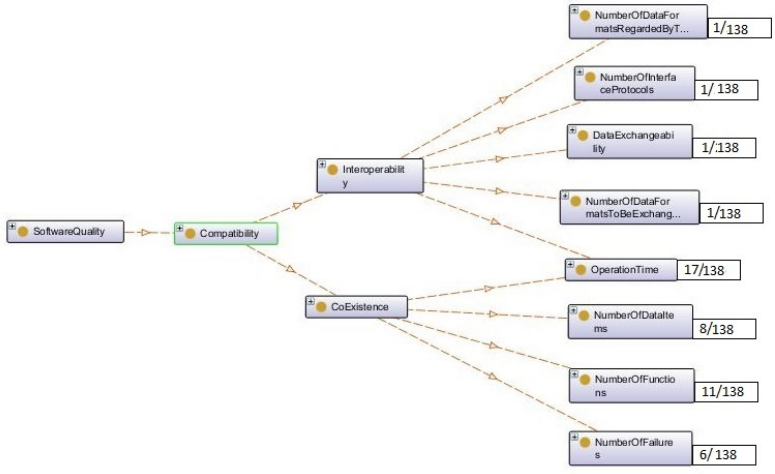

Fig. 2.16. Component of the weighted base ontology for Compatibility

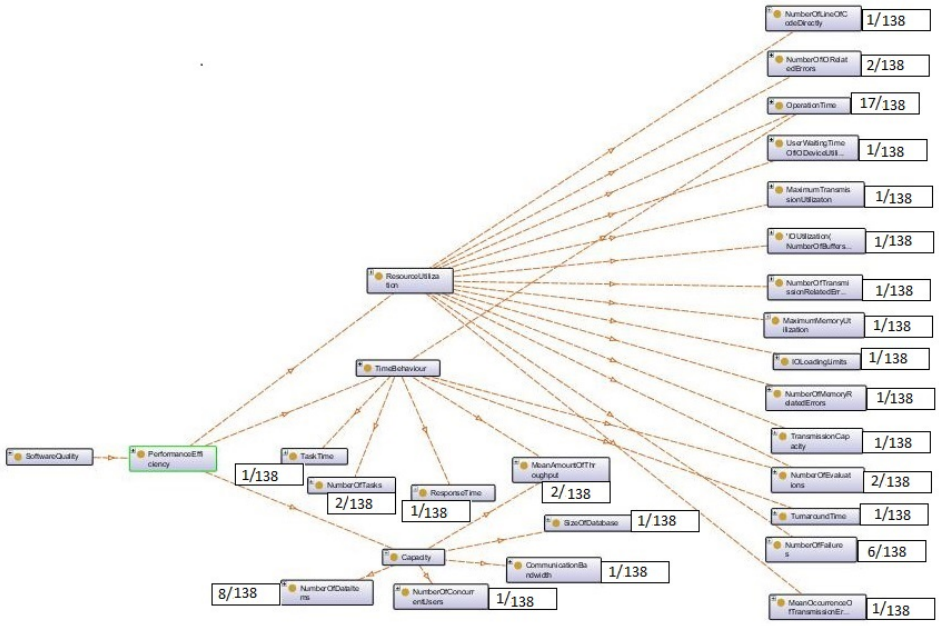

Fig. 2.17. Component of the weighted base ontology for Performance Efficiency

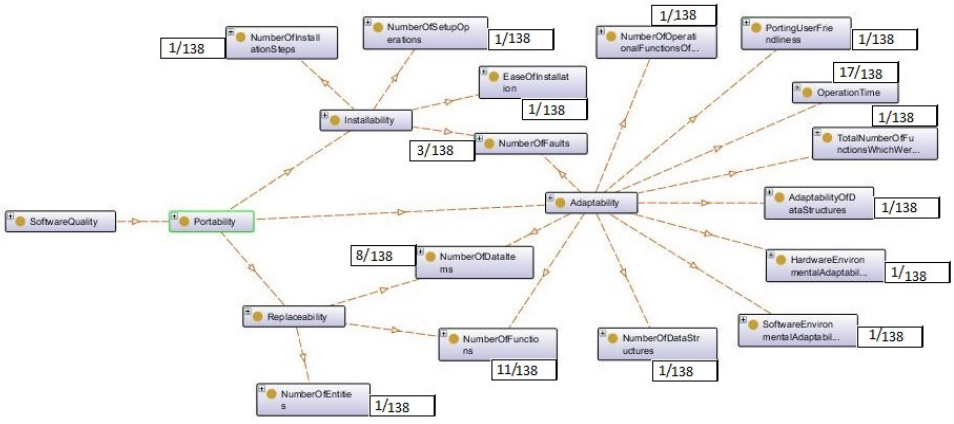

Fig. 2.18. Component of the weighted base ontology for Portability

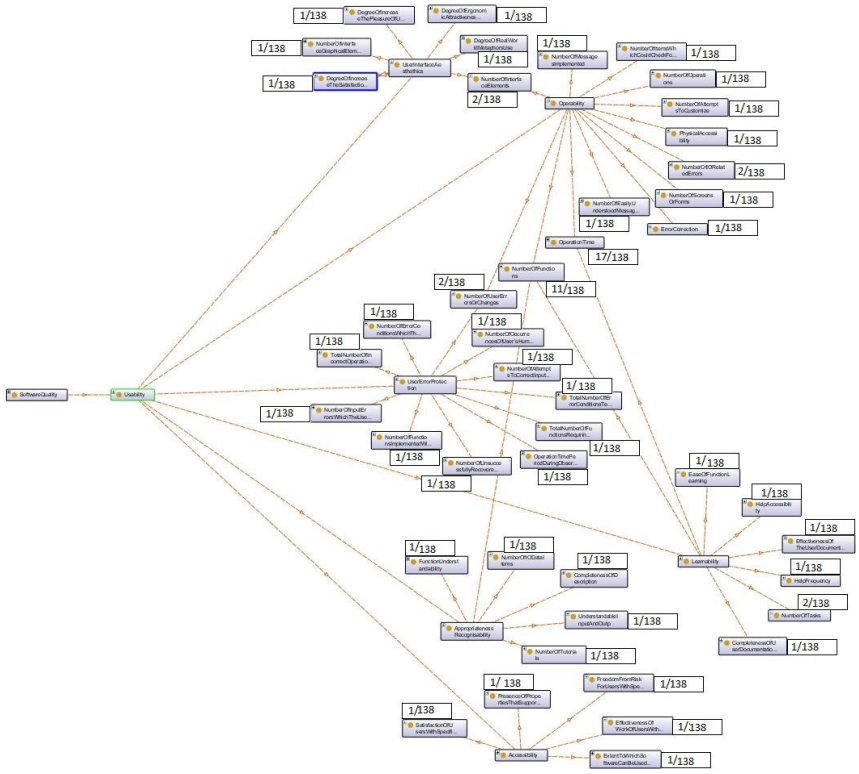

Fig. 2.19. Component of the weighted base ontology for Usability

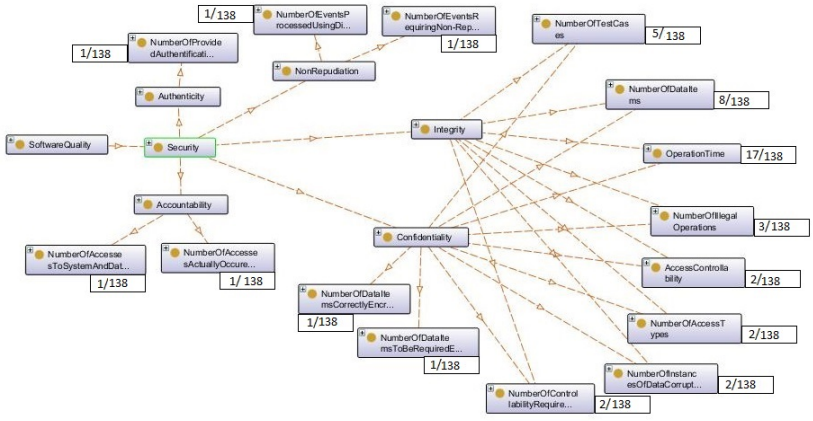

Fig. 2.20. Component of the weighted base ontology for Security

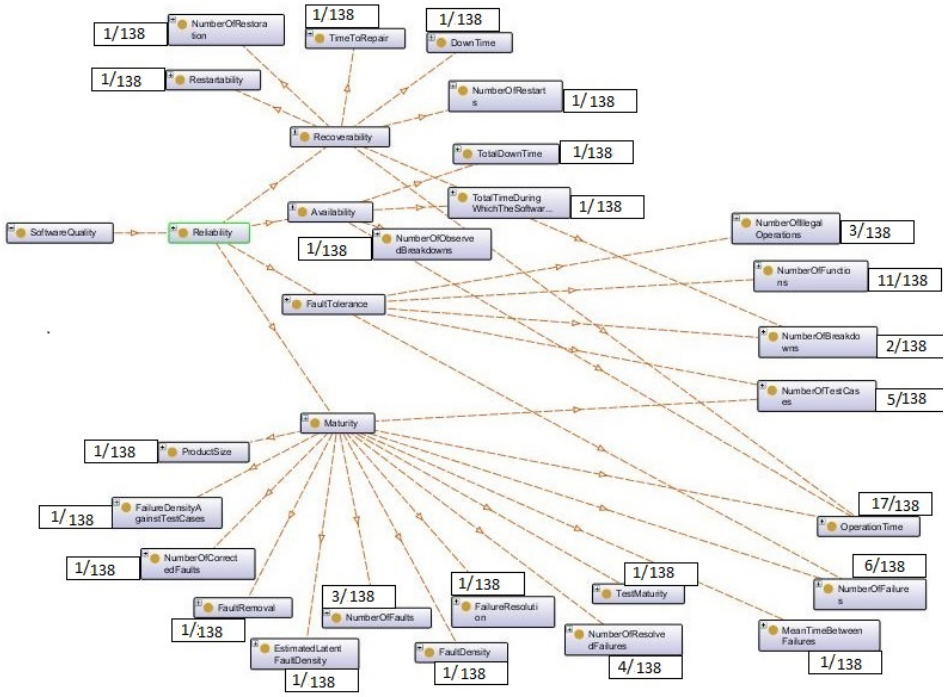

Fig. 2.21. Component of the weighted base ontology for Reliability

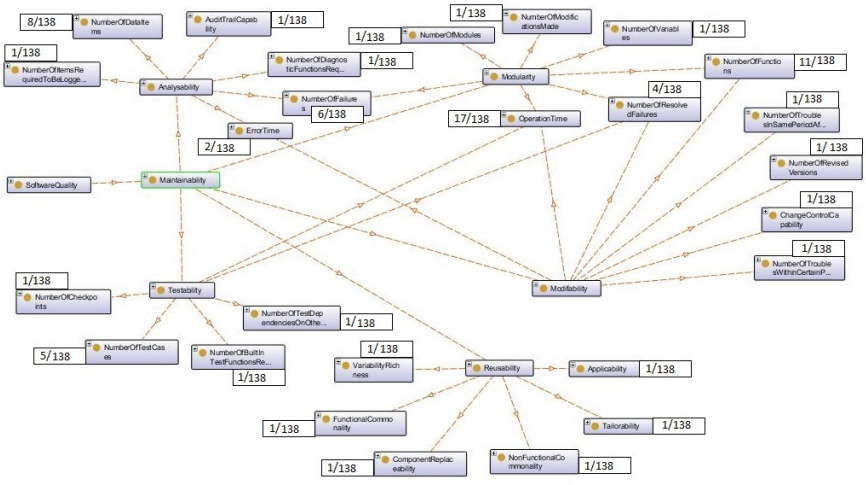

Fig. 2.22. Component of the weighted base ontology for Maintainability

Marking the weights of software quality attributes in the base ontology makes it possible to sort all missing in the specification of software requirements quality attributes in descending order of weights values, i.e. to set the priority of their addition to the specification of software requirements.

Method for assessing the sufficiency of quality information (according to ISO 25010:2011) in software requirements specifications based on weighted ontology consists of the following stages:

1) implementation of steps 1-2 of the above-described method for assessing the sufficiency of quality information (according to ISO 25010:2011) in software requirements specifications based on the ontology;

2) comparison of the obtained ontology with the weighted base ontology of the subject area "Software Engineering" (part "Software Quality"), the components of which are shown in Fig. 2.15-2.22, i.e. comparison of the set of attributes $\{qms_1,...,qms_{nm}\}$ from the ontology for determining the quality of concrete software $O_Q^{real} = \begin{Bmatrix} qch_1,...,qch_{nch}, qsch_1,...,qsch_{nsch}, qms_1,...,qms_{nm}, \\ "depends\ on", f(), f_1(),...,f_8(), \varphi_1(),...,\varphi_{31}() \end{Bmatrix}$ with the corresponding set $\{(qms_1, w_1^m),...,(qms_{138}, w_{138}^m)\}$ from the weighted base ontology of the subject area "Software Engineering" (part "Software Quality") $O_Q^w = \begin{Bmatrix} qch_1,...,qch_8, qsch_1,...,qsch_{31}, (qms_1, w_1^m),...,(qms_{138}, w_{138}^m), \\ "depends\ on", f(), f_1(),...,f_8(), \varphi_1(),...,\varphi_{31}() \end{Bmatrix}$;

3) identifying attributes that are missing in the ontology for determining the quality of concrete software, i.e. the formation of a subset

$$\{(qms_1, w_1),...,(qms_{(138-nm)}, w_{(138-nm)})\} = \{(qms_1, w_1^m),...,(qms_{138}, w_{138}^m)\} - \\ -(\{(qms_1, w_1^m),...,(qms_{138}, w_{138}^m)\} \cap \{qms_1,...,qms_{nm}\})$$

where $\{(qms_1, w_1),...,(qms_{(138-nm)}, w_{(138-nm)})\} \in SMMW$,

$\{(qms_1, w_1^m),...,(qms_{138}, w_{138}^m)\} \in O_Q^w$, $\{qms_1,...,qms_{nm}\} \in O_Q^{real}$ (if the generated set is not empty, then the information in the specification is not sufficient for calculating the quality sub-characteristics and characteristics – the more elements in the specified set, the lower the level of sufficiency of information in the requirements);

4) ordering of missing attributes in descending order of values of weights; the numerator of the weights of each missing quality attribute indicates the number of software quality sub-characteristics that cannot be calculated without this attribute;

5) identification of quality sub-characteristics and characteristics that cannot be calculated on the basis of available attributes;

6) making a decision on the need to supplement the specification of software requirements with the necessary attributes in the presence of sub-characteristics and characteristics, the value of which cannot be determined on the basis of the attributes

available in the specification; the attributes with higher weights must be first considered and included in the specification (the first in the sorted list of missing attributes);

7) forming a request to the developers of specifications for the requirements governing the attributes of quality (with the recommended priority of the supplement); making the necessary additions to the specification with quality information (attributes);

8) repeating steps 1-7 until all attributes are included in the requirements specification and until it is possible to define all sub-characteristics and quality characteristics, or until the conclusion about insufficiency of information on quality in the software requirements specification satisfies the customer.

Scheme of the method for assessing the sufficiency of quality information (according to ISO 25010:2011) in software requirements specifications based on weighted ontology is represented in Fig. 2.23 [144].

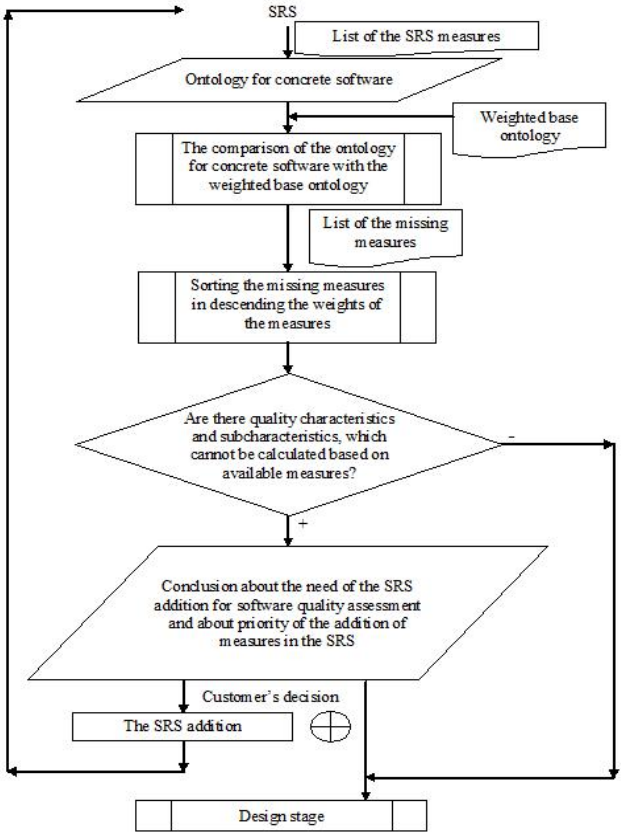

Fig. 2.23. Scheme of the method for assessing the sufficiency of quality information (according to ISO 25010:2011) in software requirements specifications based on weighted ontology [144]

Method of generating and filling the template of ontology for determining the quality of concrete software [74-76, 142-144, 146]. For assessing the sufficiency of quality information (according to ISO 25010:2011) in the specifications of software requirements by the methods for assessing the sufficiency of quality information (according to ISO 25010:2011) in the specifications of software requirements based on ontology and based on weighted ontology, it's necessary to generate and filled a template of ontology for determining the quality of concrete software. The process of generating and filling the template of ontology for determining the quality of concrete software should be automated and simplified to minimize the impact of the human factor and to simplify the assessment of quality information sufficiency (ISO 25010:2011) in software requirements specifications by both developer and customer.

Based on the developed base (universal) ontological model of the subject area "Software Engineering" (part "Specification of software requirements (quality attributes)"), the base ontology of the subject area "Software Engineering" (part "Specification of software requirements (quality attributes)") was developed – Fig. 2.24. In this ontology, the required quality attributes are presented taking into account the distribution by sections of the specification, due to which the developed ontology is a template of specification of software requirements in terms of attributes and provides visual hints to the user about the location of certain attributes in the software requirements specification.

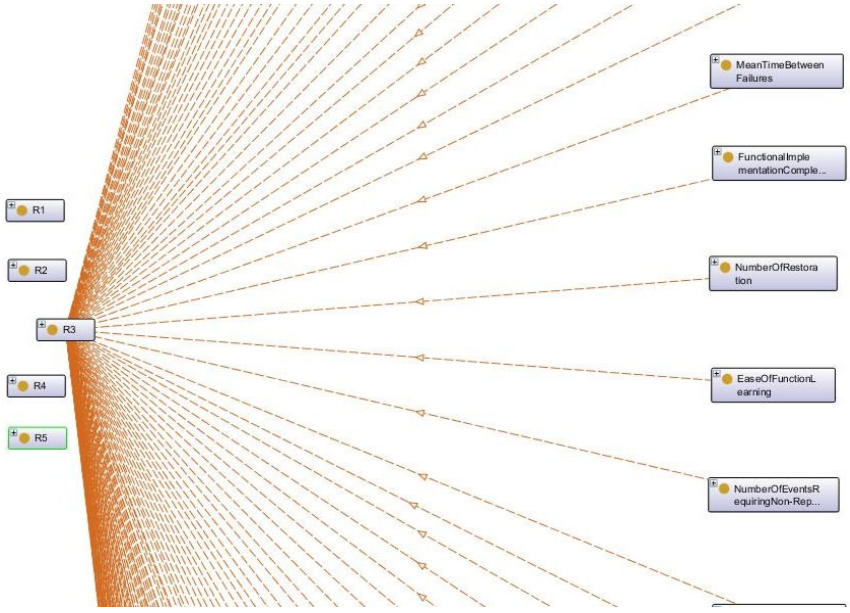

Fig. 2.24. Fragment of the ontology of the subject area "Software Engineering" (part "Specification of software requirements (quality attributes)") – a template of specification of software requirements in terms of the presence of quality attributes [146]

The developed base ontology of the subject area "Software Engineering" (part "Software Quality"), presented in Fig. 2.6-2.13, contains all 138 software quality attributes that may be contained in the software requirements specification. Therefore, in order to automate and simplify the generation and filling of the template of ontology for determining the quality of concrete software, we develop a method of generating and filling the template of ontology for determining the quality of concrete software based on the base ontology of the subject area "Software Engineering" (part "Software Quality").

The method of generating and filling the template of ontology for determining the quality of concrete software consists of the following steps:

1) to open the base ontology of the subject area "Software Engineering" (part "Software Quality");

2) to remove from the base ontology all the attributes that were not identified in the specification of the software requirements of the concrete software, using the results of the analysis of the specification of software requirements of the concrete software for the presence of attributes, which are necessary for determining the software quality sub-characteristics and characteristics, performed in step №1 of the method for assessing the sufficiency of quality information (according to ISO 25010:2011) in software requirements specifications based on ontology;

3) to save the changes, thus creating an ontology for determining the quality of concrete software.

Method of forming a logical conclusion about the sufficiency of information on quality (according to ISO 25010:2011) in the specifications of software requirements [74-76, 142-144, 146]. First of all, we develop *production rules for forming a logical conclusion about the sufficiency of quality information (according to ISO 25010:2011) in the software requirements specifications* (set $PR = \{ pr_1, .., pr_{140} \}$) based on the base and weighted base ontologies of the subject area Software Engineering (part "Software Quality"):

1) if the software requirements specification does not have the "Number Of Functions" attribute (user removed it from the base ontology), then: $fc := fc + 1$ (there is insufficient information for determining the Functional Completeness sub-characteristic, so the counter of missing attributes for this sub-characteristic is increased by 1), $fa := fa + 1$ (counter of missing attributes for Functional Appropriateness sub-characteristic), $ft := ft + 1$ (counter of missing attributes for Fault Tolerance), $ar := ar + 1$ (counter of missing attributes for Appropriateness Recognisability), $lb := lb + 1$ (counter of missing attributes for Learnability), $ob := ob + 1$ (counter of missing attributes for Operability), $md := md + 1$ (counter of missing attributes for Modularity), $mfb := mfb + 1$ (counter of missing attributes for Modifability), $cex := cex + 1$ (counter of missing attributes for CoExistence), $ab := ab + 1$ (counter of missing attributes for Adaptability), $rb := rb + 1$ (counter of missing attributes for Replaceability) and, accordingly, $fy := fy + 2$ (there is insufficient information for determining the two sub-characteristics of the Functional Suitability characteristic, so the

counter of missing attributes for this characteristic is increased by 2), $ry := ry + 1$ (counter of missing attributes for Reliability characteristic), $uy := uy + 3$ (counter of missing attributes for Usability), $my := my + 2$ (counter of missing attributes for Maintainability), $cy := cy + 1$ (counter of missing attributes for Compatibility), $py := py + 2$ (counter of missing attributes for Portability), i.e. there is insufficient information for determining 11 of the 31 sub-characteristics, as well as 6 of the 8 software quality characteristics; `mas[Number Of Functions]:=11/138` (in the corresponding element of the array `mas` the weight of the missing attribute is written, determined by the weighted base ontology of the subject area "Software Engineering (part" Software Quality")");

2) if the software requirements specification does not have the "Operation Time" attribute, then: $fcr := fcr + 1$ (counter of missing attributes for Functional Correctness sub-characteristic), $fa := fa + 1$, $ma := ma + 1$ (counter of missing attributes for Maturity), $av := av + 1$ (counter of missing attributes for Availability), $rvb := rvb + 1$ (counter of missing attributes for Recoverability), $tb := tb + 1$ (counter of missing attributes for Time Behaviour), $ru := ru + 1$ (counter of missing attributes for Resource Utilization), $lb := lb + 1$, $ob := ob + 1$, $md := md + 1$, $mfb := mfb + 1$, $tst := tst + 1$ (counter of missing attributes for Testability), $cf := cf + 1$ (counter of missing attributes for Confidentiality), $ig := ig + 1$ (counter of missing attributes for Integrity), $cex := cex + 1$, $io := io + 1$ (counter of missing attributes for Interoperability), $ab := ab + 1$ and, accordingly, $fy := fy + 2$, $ry := ry + 3$, $ey := ey + 2$ (counter of missing attributes for Performance Efficiency characteristic), $uy := uy + 2$, $my := my + 3$, $sy := sy + 2$ (counter of missing attributes for Security), $cy := cy + 2$, $py := py + 1$, i.e. there is insufficient information for determining 17 of the 31 sub-characteristics, as well as 8 of the 8 software quality characteristics; `mas[Operation Time]:=17/138`;

3)–138) – rules for the other 136 attributes are similarly formed;

139) if $fc = 0$ and $fcr = 0$ and $fa = 0$ and $ma = 0$ and $av = 0$ and $ft = 0$ and $rvb = 0$ and $tb = 0$ and $ru = 0$ and $ccy = 0$ and $ar = 0$ and $lb = 0$ and $ob = 0$ and $uepr = 0$ and $uiae = 0$ and $acsb = 0$ and $md = 0$ and $rusb = 0$ and $anb = 0$ and $mfb = 0$ and $tsb = 0$ and $cf = 0$ and $ig = 0$ and $nrpd = 0$ and $accb = 0$ and $athc = 0$ and $cex = 0$ and $io = 0$ and $ab = 0$ and $ib = 0$ and $rb = 0$, then in the specification of software requirements, there is sufficient information for determining all sub-characteristics, else: in the specification of software requirements there is insufficient information on quality (attributes) for determining the certain sub-characteristics of software quality:

- if $0 < fc \le 4$, then: in the specification of software requirements there is insufficient information for determining the sub-characteristic Functional Completeness; if $fc = 4$, then

there is no information in the specification of software requirements for determining the sub-characteristic Functional Completeness;

- if $0 < fcr \le 5$, then: there is insufficient information for Functional Correctness; if $fcr = 5$, then there is no information for Functional Correctness;

- if $0 < fa \le 6$, then: there is insufficient information for Functional Appropriateness; if $fa = 6$, then there is no information for Functional Appropriateness;

- if $0 < ma \le 14$, then: there is insufficient information for Maturity; if $ma = 14$, then there is no information for Maturity;

- if $0 < av \le 4$, then: there is insufficient information for Availability; if $av = 4$, then there is no information for Availability;

- if $0 < ft \le 5$, then: there is insufficient information for Fault Tolerance; if $ft = 5$, then there is no information for Fault Tolerance;

- if $0 < rvb \le 7$, then: there is insufficient information for Recoverability; if $rvb = 7$, then there is no information for Recoverability;

- if $0 < tb \le 7$, then: there is insufficient information for Time Behaviour; if $tb = 7$, then there is no information for Time Behaviour;

- if $0 < ru \le 14$, then: there is insufficient information for Resource Utilization; if $ru = 14$, then there is no information for Resource Utilization;

- if $0 < ccy \le 5$, then: there is insufficient information for Capacity; if $ccy = 5$, then there is no information for Capacity;

- if $0 < ar \le 6$, then: there is insufficient information for Appropriateness Recognisability; if $ar = 6$, then there is no information for Appropriateness Recognisability;

- if $0 < lb \le 8$, then: there is insufficient information for Learnability; if $lb = 8$, then there is no information for Learnability;

- if $0 < ob \le 13$, then: there is insufficient information for Operability; if $ob = 13$, then there is no information for Operability;

- if $0 < uepr \le 11$, then: there is insufficient information for User Error Protection; if $uepr = 11$, then there is no information for User Error Protection;

- if $0 < uiae \le 6$, then: there is insufficient information for User Interface Aesthethics; if $uiae = 6$, then there is no information for User Interface Aesthethics;

- if $0 < acsb \le 5$, then: there is insufficient information for Accessibility; if $acsb = 5$, then there is no information for Accessibility;

- if $0 < md \le 7$, then: there is insufficient information for Modularity; if $md = 7$, then there is no information for Modularity;

- if $0 < rusb \le 6$, then: there is insufficient information for Reusability; if $rusb = 6$, then there is no information for Reusability;

- if $0 < anb \le 6$, then: there is insufficient information for Analysability; if $anb = 6$, then there is no information for Analysability;

- if $0 < mfb \le 8$, then: there is insufficient information for Modifability; if $mfb = 8$, then there is no information for Modifability;

- if $0 < tst \le 6$, then: there is insufficient information for Testability; if $tst = 6$, then there is no information for Testability;

- if $0 < cf \le 10$, then: there is insufficient information for Confidentiality; if $cf = 10$, then there is no information for Confidentiality;

- if $0 < ig \le 8$, then: there is insufficient information for Integrity; if $ig = 8$, then there is no information for Integrity;

- if $0 < nrpd \le 2$, then: there is insufficient information for Non Repudiation; if $nrpd = 2$, then there is no information for Non Repudiation;

- if $0 < athc \le 1$, then: there is insufficient information for Authenticity; if $athc = 1$, then there is no information for Authenticity;

- if $0 < accb \le 2$, then: there is insufficient information for Accountability; if $accb = 2$, then there is no information for Accountability;

- if $0 < cex \le 4$, then: there is insufficient information for CoExistence; if $cex = 4$, then there is no information for CoExistence;

- if $0 < io \le 5$, then: there is insufficient information for Interoperability; if $io = 5$, then there is no information for Interoperability;

- if $0 < ab \le 11$, then: there is insufficient information for Adaptability; if $ab = 11$, then there is no information for Adaptability;

- if $0 < ib \le 4$, then: there is insufficient information for Installability; if $ib = 4$, then there is no information for Installability;

- if $0 < rb \le 3$, then: there is insufficient information for Replaceability; if $rb = 3$, then there is no information for Replaceability;

140) if $fy = 0$ and $ry = 0$ and $ey = 0$ and $uy = 0$ and $my = 0$ and $sy = 0$ and $cy = 0$ and $py = 0$, then the information on the quality (attributes) in the specification of software requirements is sufficient for determining the software quality characteristics according to ISO 25010:2011, else:

1. there is insufficient quality information (attributes) in the software requirements specification for determining the certain software quality characteristics:

- if $0 < fy \le 15$, then: there is insufficient information for determining Functional Suitability characteristic in the specification of software requirements; if $fy = 15$, then there is no information for determining Functional Suitability characteristic in the software requirements specification;

- if $0 < ry \le 30$, then: there is insufficient information for Reliability; if $ry = 30$, then there is no information for Reliability;

- if $0 < ey \le 26$, then: there is insufficient information for Performance Efficiency; if $ey = 26$, then there is no information for Performance Efficiency;

- if $0 < uy \le 49$, then: there is insufficient information for Usability; if $uy = 49$, then there is no information for Usability;

- if $0 < my \le 33$, then: there is insufficient information for Maintainability; if $my = 33$, then there is no information for Maintainability;

- if $0 < sy \le 23$, then: there is insufficient information for Security; if $sy = 23$, then there is no information for Security;

- if $0 < cy \le 9$, then: there is insufficient information for Compatibility; if $cy = 9$, then there is no information for Compatibility;

- if $0 < py \le 18$, then: there is insufficient information for Portability; if $py = 18$, then there is no information for Portability;

2. to sort array `mas` in descending order of element values (weights of missing attributes);

3. to display the indices of those elements of the sorted array `mas`, which have a value other than 0 – as the recommended priority of adding attributes to the specification of software requirements.

Based on the production rules for forming a logical conclusion about the sufficiency of quality information (according to ISO 25010:2011) in the software requirements specifications, we develop *the method of forming a logical conclusion about the sufficiency of information on quality (according to ISO 25010:2011) in the specifications of software requirements*:

1) according to the Method for assessing the sufficiency of quality information (according to ISO 25010:2011) in software requirements specifications based on ontology, taking into account the comparative analysis of ontologies, the formation of a set of the attributes $\{qms_1,...,qms_{(138-nm)}\}$ is performed, which are missing in the ontology for determining the quality of concrete software;

2) according to the method of searching in the width in the forward direction, in the subset of production rules $\{pr_1,..., pr_{138}\}$ there is a search for a rule for each element of the set $\{qms_1,...,qms_{(138-nm)}\}$, according to which the counters of missing attributes for sub-characteristics and characteristics are calculated;

3) according to the rules from the subset $\{pr_{139}, pr_{140}\}$, the analysis of information on quality (attributes) in the specification of software requirements for sufficiency is performed, and, in case of insufficiency, conclusions are formed, for determining of which software quality sub-characteristics and characteristics the information is insufficient, as well as the sorted (by weights) list of missing attributes is formed as a recommended priority of adding attributes to the software requirements specification;

4) the numerical assessment of the sufficiency of the volume of quality information (attributes), available in the requirements, is performed – should strive for this assessment to be as large as possible, i.e. to go to 1:

- by sub-characteristics:

$$q_{schr} = \frac{fc}{4} + \frac{fcr}{5} + \frac{fa}{6} + \frac{ma}{14} + \frac{av}{4} + \frac{ft}{5} + \frac{rvb}{7} + \frac{tb}{7} + \frac{ru}{14} + \frac{ccy}{5} + \\ + \frac{ar}{6} + \frac{lb}{8} + \frac{ob}{13} + \frac{uepr}{11} + \frac{uiae}{6} + \frac{acsb}{5} + \frac{md}{7} + \frac{rusb}{6} + \frac{anb}{6} + \frac{mfb}{8} + , \\ + \frac{tst}{6} + \frac{cf}{10} + \frac{ig}{8} + \frac{nrpd}{2} + \frac{accb}{2} + \frac{athc}{1} + \frac{cex}{4} + \frac{io}{5} + \frac{ab}{11} + \frac{ib}{4} + \frac{rb}{3} \quad (2.20)$$

where q_{schr} – the number of sub-characteristics that cannot be calculated based on the attributes, available in the specification; numbers in fraction numerators indicate the number of attributes missing in the specification for a certain quality sub-characteristic, numbers in fraction denominators indicate the number of required attributes for each quality sub-characteristic;

$$D_{schr} = \frac{31 - q_{schr}}{31}, \quad (2.21)$$

where D_{schr} – numerical assessment of the sufficiency of the amount of information (attributes) available in the specification for assessing the quality of software by sub-characteristics;

- by characteristics (taking into account the numerical assessment of the sufficiency of the amount of quality information (attributes), available in the specification of requirements):

$$q_{chr} = \frac{fy}{15} + \frac{ry}{30} + \frac{ey}{26} + \frac{uy}{49} + \frac{my}{33} + \frac{sy}{23} + \frac{cy}{9} + \frac{py}{18}, \quad (2.22)$$

where q_{chr} – the number of characteristics that cannot be calculated based on the attributes, available in the specification; numbers in fraction numerators indicate the number of attributes missing in the specification for a certain quality characteristic, numbers in fraction denominators indicate the number of required attributes for each quality characteristic;

$$D_{chr} = \frac{8 - q_{chr}}{8}, \quad (2.23)$$

where D_{chr} – numerical assessment of the sufficiency of the amount of information (attributes) available in the specification for assessing the quality of software by characteristics;

5) numerical assessment of the sufficiency of the available (after supplementing) quality information (attributes) in the software requirements specification (taking into account that the methods for assessing the sufficiency of quality information (according to ISO 25010:2011) in the software requirements specifications based on ontology and based on weighted ontology are iterative, after forming a conclusion about the insufficient information in the specification, the specification of requirements will be supplemented with

the necessary attributes, which will increase the sufficiency of information on quality in the requirements specification):

- by sub-characteristics:

$$q'_{schr} = \frac{fc'}{4} + \frac{fcr'}{5} + \frac{fa'}{6} + \frac{ma'}{14} + \frac{av'}{4} + \frac{ft'}{5} + \frac{rvb'}{7} + \frac{tb'}{7} + \frac{ru'}{14} + \frac{ccy'}{5} + \\ + \frac{ar'}{6} + \frac{lb'}{8} + \frac{ob'}{13} + \frac{uepr'}{11} + \frac{uiae'}{6} + \frac{acsb'}{5} + \frac{md'}{7} + \frac{rusb'}{6} + \frac{anb'}{6} + \frac{mfb'}{8} + , \\ + \frac{tst'}{6} + \frac{cf'}{10} + \frac{ig'}{8} + \frac{nrpd'}{2} + \frac{accb'}{2} + \frac{athc'}{1} + \frac{cex'}{4} + \frac{io'}{5} + \frac{ab'}{11} + \frac{ib'}{4} + \frac{rb'}{3} \quad (2.24)$$

where q'_{schr} – the number of sub-characteristics that cannot be calculated based on the attributes, available in the specification after specification's supplementing; numbers in fraction numerators indicate the number of attributes missing in the specification for a certain quality sub-characteristic (after supplementing);

$$D'_{schr} = \frac{31 - q'_{schr}}{31}, \quad (2.25)$$

where D'_{schr} – numerical assessment of the sufficiency of the amount of information (attributes) available in the specification (after supplementing) for assessing the quality of software by sub-characteristics;

- by characteristics:

$$q'_{chr} = \frac{fy'}{15} + \frac{ry'}{30} + \frac{ey'}{26} + \frac{uy'}{49} + \frac{my'}{33} + \frac{sy'}{23} + \frac{cy'}{9} + \frac{py'}{18}, \quad (2.26)$$

where q'_{chr} – the number of characteristics that cannot be calculated based on the attributes, available in the specification after specification's supplementing; numbers in fraction numerators indicate the number of attributes missing in the specification for a certain quality characteristic (after supplementing);

$$D'_{chr} = \frac{8 - q'_{chr}}{8}, \quad (2.27)$$

where D'_{chr} – numerical assessment of the sufficiency of the amount of information (attributes) available in the specification (after supplementing) for assessing the quality of software by characteristics;

6) the gain of the sufficiency of the amount of information on quality (attributes) in the specification of software requirements (after supplementing the specification with the necessary attributes) is calculated:

- by sub-characteristics:

$$\Delta q_{schr} = q_{schr} - q'_{schr} = (\frac{fc}{4} - \frac{fc'}{4}) + (\frac{fcr}{5} - \frac{fcr'}{5}) + ... + (\frac{rb}{3} - \frac{rb'}{3}), \quad (2.28)$$

where Δq_{schr} – the number of sub-characteristics that became possible to calculate after the supplementing the specification with the attributes;

$$\Delta D_{schr} = D'_{schr} - D_{schr} = \frac{\Delta q_{schr}}{31}, \quad (2.29)$$

where ΔD_{schr} – the gain of the sufficiency of the amount of information on quality (attributes) in the specification of software requirements (after supplementing the specification with the necessary attributes) for assessing the quality of software by sub-characteristics;

- by characteristics:

$$\Delta q_{chr} = q_{chr} - q'_{chr} = (\frac{fy}{15} - \frac{fy'}{15}) + (\frac{ry}{30} - \frac{ry'}{30}) + ... + (\frac{py}{18} - \frac{py'}{18}), \quad (2.30)$$

where Δq_{chr} – the number of characteristics that became possible to calculate after the supplementing the specification with the attributes;

$$\Delta D_{chr} = D'_{chr} - D_{chr} = \frac{\Delta q_{chr}}{8}, \quad (2.31)$$

where ΔD_{chr} – the gain of the sufficiency of the amount of information on quality (attributes) in the specification of software requirements (after supplementing the specification with the necessary attributes) for assessing the quality of software by characteristics.

The scheme of developed method of forming a logical conclusion about the sufficiency of information on quality (according to ISO 25010:2011) in the specifications of requirements is represented on Fig. 2.25.

2.3. Methods for Assessing the Sufficiency of Quality Information (for Metric Analysis) in Software Requirements Specifications

Method for assessing the sufficiency of quality information (for metric analysis) in software requirements specifications based on ontology [74-76, 142-144]. For assessing the sufficiency of quality information (for metric analysis) in the software requirements specifications, the method based on comparative analysis of the ontologies is proposed.

First of all, for the subject area "Software Engineering" (part “Software Quality. Metric analysis”) we develop a base ontology, which contains information on the software project complexity, software complexity, software project quality, software quality. These

characteristics are determined on the basis of metrics selected in terms of the possibility of their calculation at the design stage of the architecture, which are calculated on the basis of quality indicators. The concept of the base ontology of the subject area "Software Engineering" (part "Software Quality. Metric analysis") is presented in Fig. 2.26.

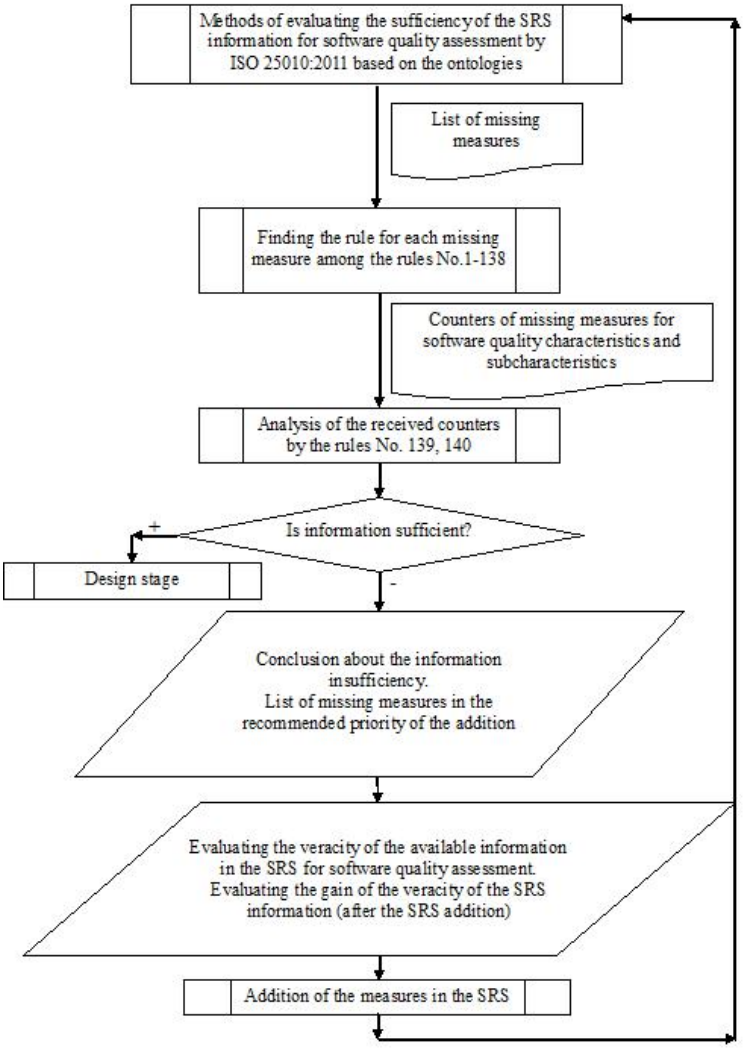

Fig. 2.25. The scheme of developed method of forming a logical conclusion about the sufficiency of information on quality (according to ISO 25010:2011) in the specifications of requirements [144]

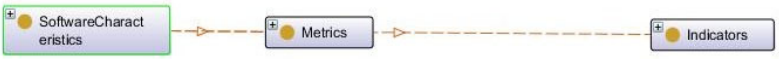

Fig. 2.26. Concept of the base ontology of the subject area "Software Engineering" (part "Software Quality. Metric analysis")

The base ontology of the subject area "Software Engineering" (part "Software Quality. Metric analysis") is formed by components for:

1) software project complexity (Fig. 2.27);
2) software complexity (Fig. 2.28);
3) software project quality (Fig. 2.29);
4) software quality (Fig. 2.30).

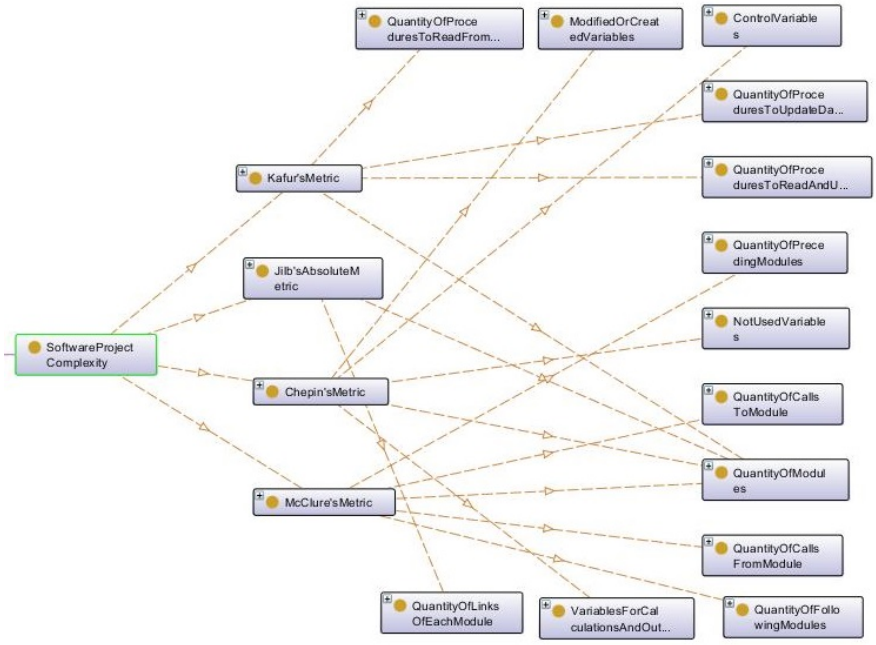

Fig. 2.27. Component of base ontology for software project complexity based on the metric analysis

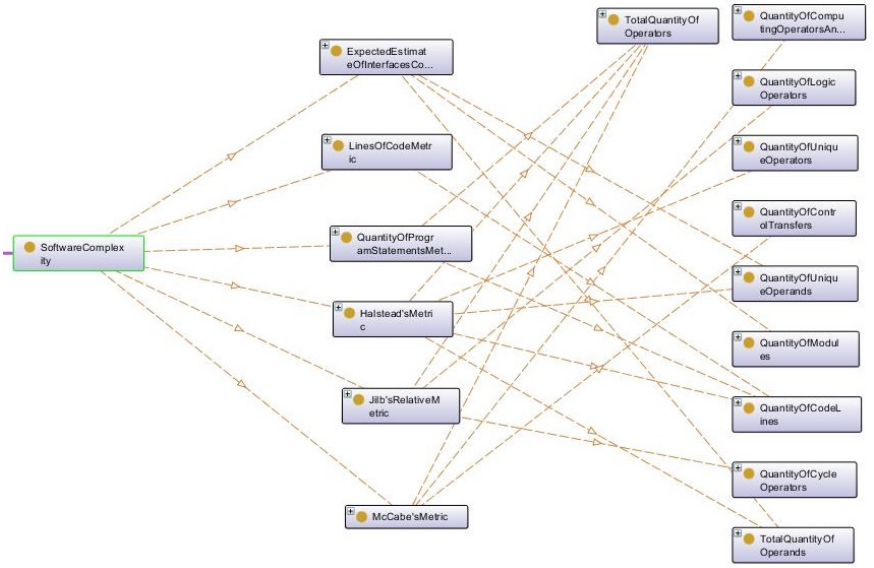

Fig. 2.28. Component of base ontology for software complexity based on the metric analysis

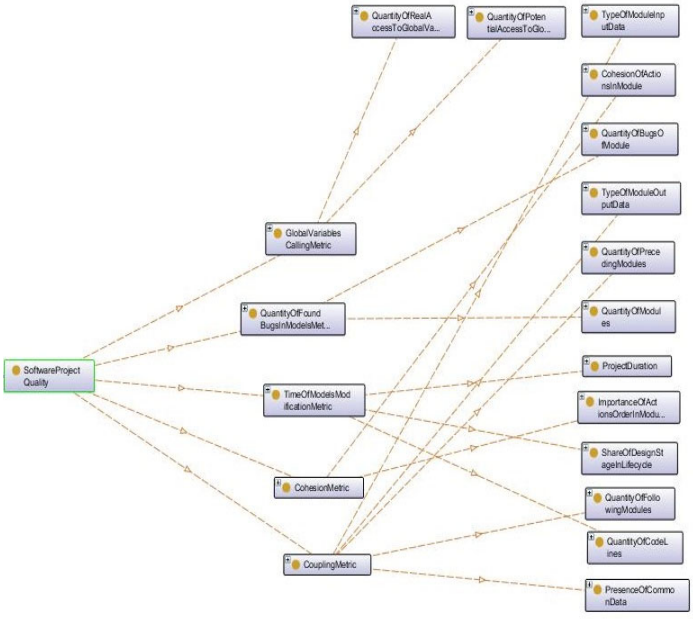

Fig. 2.29. Component of base ontology for software project quality based on the metric analysis

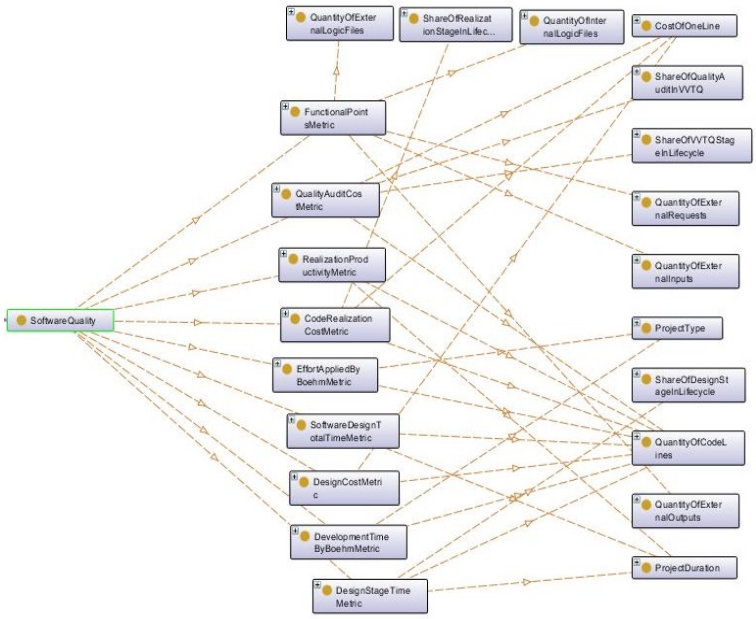

Fig. 2.30. Component of base ontology for software quality based on the metric analysis

Method for assessing the sufficiency of quality information (for metric analysis) in software requirements specifications based on ontology consists of the following stages:

1) analysis of sections of the specification of requirements for the concrete software for the presence of indicators necessary to determine the software quality and complexity metrics, i.e. for the presence of elements of the subset $\{sqcxi_1,...,sqcxi_{nqcxi}\} \in O_{metr}^{real}$;

2) generating and filling the template of ontology for determining the quality of concrete software (based on the metric analysis), which is formed by components for software project complexity, software complexity, software project quality, software quality of the concrete software, i.e. generating and filling the template of ontology $O_{metr}^{real} = \begin{Bmatrix} sqm_1,...,sqm_{nqm}, scxm_1,...,scxm_{ncxm}, sqcxi_1,...,sqcxi_{nqcxi}, \\ "depends\ \ on", \psi(), \psi_1(),...,\psi_{14}(), \phi(), \phi_1(),...,\phi_{10}() \end{Bmatrix}$;

3) comparison of the obtained ontology with the base ontology of the subject area "Software Engineering" (part "Software Quality.Metric analysis"), the components of which are shown in Fig. 2.27-2.30, i.e. comparison of the set of indicators $\{sqcxi_1,...,sqcxi_{ni}\}$ from the ontology for determining the quality of concrete software (based on the metric

information) $O^{real}_{metr} = \begin{Bmatrix} sqm_1,...,sqm_{nqm}, scxm_1,...,scxm_{ncxm}, sqcxi_1,...,sqcxi_{nqcxi}, \\ "depends\ on", \psi(), \psi_1(),...,\psi_{14}(), \phi(), \phi_1(),...,\phi_{10}() \end{Bmatrix}$ with the corresponding set $\{ sqcxi_1,...,sqcxi_{42} \}$ from the base ontology of the subject area "Software Engineering" (part "Software Quality.Metric analysis") $O_{metr} = \begin{Bmatrix} sqm_1,...,sqm_{14}, scxm_1,...,scxm_{10}, sqcxi_1,...,sqcxi_{42}, \\ "depends\ on", \psi(), \psi_1(),...,\psi_{14}(), \phi(), \phi_1(),...,\phi_{10}() \end{Bmatrix}$;

4) identifying indicators that are missing in the ontology for determining the quality of concrete software (based on the metric information), i.e. the formation of a set

$$\{ sqcxi_1,...,sqcxi_{(42-nqcxi)} \} = \\ = \{ sqcxi_1,...,sqcxi_{42} \} - (\{ sqcxi_1,...,sqcxi_{42} \} \cap \{ sqcxi_1,...,sqcxi_{nqcxi} \}),$$

where $\{ sqcxi_1,...,sqcxi_{(42-nqcxi)} \} \in SMIQM$, $\{ sqcxi_1,...,sqcxi_{42} \} \in O_{metr}$, $\{ sqcxi_1,...,sqcxi_{nqcxi} \} \in O^{real}_{metr}$ (if the generated set is not empty, then the information in the specification is not sufficient for calculating the software metrics – the more elements in the specified set, the lower the level of sufficiency of information in the requirements);

5) identification (on the basis of comparative analysis of the developed ontologies) of metrics that cannot be calculated on the basis of available indicators, i.e. formation of subsets $\{ sqm_1,...,sqm_{(14-nqm)} \} \in SMIQM$, $\{ scxm_1,...,scxm_{(10-ncxm)} \} \in SMIQM$;

6) identification (on the basis of comparative analysis of the developed ontologies) of software characteristics (software project complexity and quality, software complexity and quality) that cannot be calculated on the basis of metrics, which can be calculated using the available indicators;

7) the presence of software characteristics, the value of which cannot be determined on the basis of indicators, which are available in the specification of software requirements, indicates the insufficiency of information for reliable assessment of software quality based on the metric analysis, i.e. the need to supplement this specification with indicators, which are needed for calculating the one or another characteristic;

8) forming a request to the developers of specifications for the requirements governing the indicators of quality; making the necessary additions to the specification with quality information (indicators);

9) repeating steps 1-8 until all indicators are included in the requirements specification and until it is possible to define all software characteristics, or until the conclusion about insufficiency of information on quality in the software requirements specification satisfies the customer.

The final conclusion about the insufficiency of quality information (indicators) in the specification of software requirements will be taken if the cost of finalizing the specification of software requirements becomes greater than the expected effect on quality assessment.

Method for assessing the sufficiency of quality information (for metric analysis) in

software requirements specifications based on weighted ontology [74-76, 142-144]. When assessing the quality of software based on the metric analysis for ensuring the appropriate level of veracity, it is important to meet the presence in the specification of software requirements for those indicators that have higher weights.

Based on the base ontology of the subject area "Software Engineering" (part "Software Quality. Metric analysis"), the components of which are presented in Fig. 2.27- 2.30, we construct the weighted base ontology, which contains information on the weights of software quality indicators, components of which are represented in Fig. 2.31-2.34. The weighted ontology contains information on the weights of the indicators of the software requirements specification required to calculate the software complexity and quality metrics.

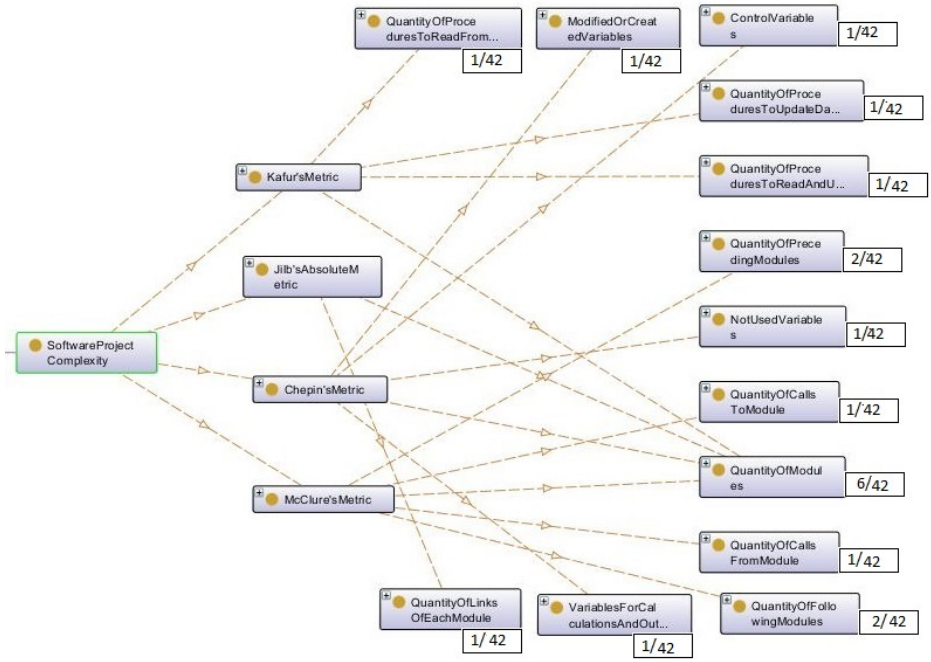

Fig. 2.31. Component of the weighted base ontology for software project complexity

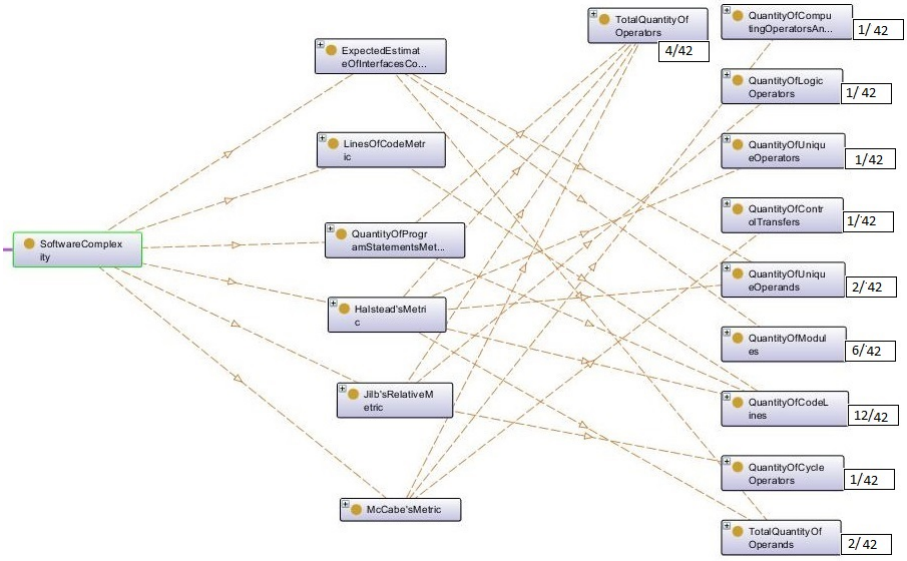

Fig. 2.32. Component of the weighted base ontology for software complexity

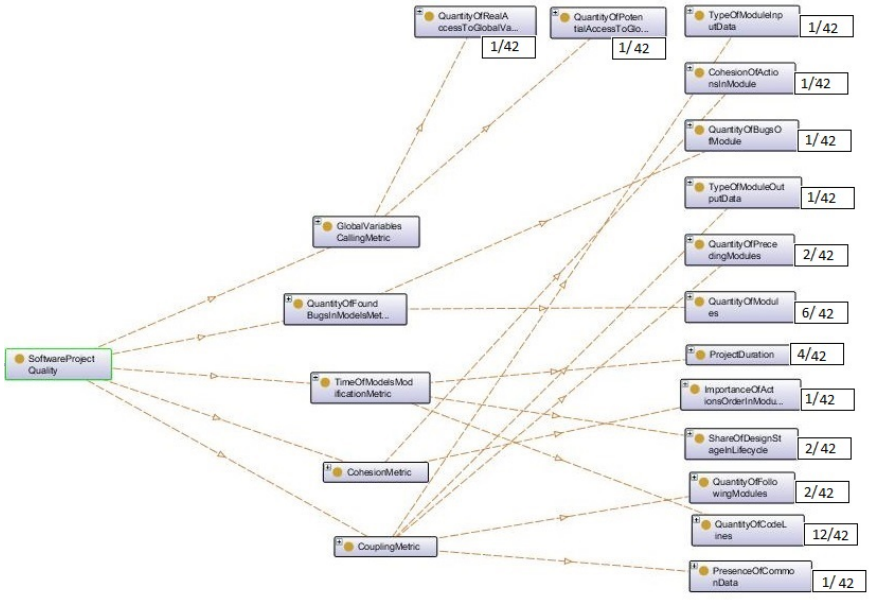

Fig. 2.33. Component of the weighted base ontology for software project quality

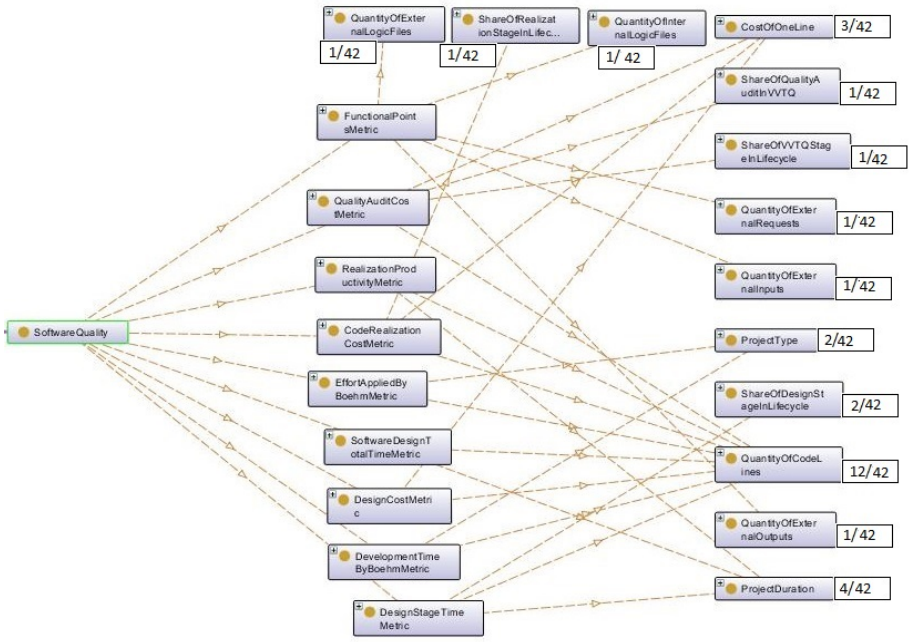

Fig. 2.34. Component of the weighted base ontology for software quality

Marking the weights of software quality indicators in the base ontology makes it possible to sort all missing in the specification of software requirements quality indicators in descending order of weights values, i.e. to set the priority of their addition to the specification of software requirements.

Similar to the developed method of assessing the sufficiency of information on quality (according to ISO 25010:2011) in the specifications of software requirements based on a weighted ontology, we propose *method for assessing the sufficiency of quality information (for metric analysis) in software requirements specifications based on weighted ontology*:

1) implementation of steps 1-2 of the above-described method for assessing the sufficiency of quality information (for metric analysis) in software requirements specifications based on the ontology;

2) comparison of the obtained ontology with the weighted base ontology of the subject area "Software Engineering" (part "Software Quality. Metric analysis"), the components of which are shown in Fig. 2.31-2.34, i.e. comparison of the set of indicators $\{sqcxi_1,...,sqcxi_{nqcxi}\}$ from the ontology for determining the quality of software (based on the metric information) $O^{real}_{metr} = \begin{Bmatrix} sqm_1,...,sqm_{nqm}, scxm_1,...,scxm_{ncxm}, sqcxi_1,...,sqcxi_{nqcxi}, \\ "depends\ on", \psi(), \psi_1(),...,\psi_{14}(), \phi(), \phi_1(),...,\phi_{10}() \end{Bmatrix}$ with

the corresponding set $\{(sqcxi_1, w_1^{ind}),...,(sqcxi_{42}, w_{42}^{ind})\}$ from the weighted base ontology of the subject area "Software Engineering" (part "Software Quality. Metric information") $O_{metr}^{w} = \begin{Bmatrix} sqm_1,...,sqm_{14}, scxm_1,...,scxm_{10}, (sqcxi_1, w_1^{ind}),...,(sqcxi_{42}, w_{42}^{ind}), \\ "depends\ on", \psi(), \psi_1(),..., \psi_{14}(), \phi(), \phi_1(),..., \phi_{10}() \end{Bmatrix}$;

3) identifying indicators that are missing in the ontology for determining the quality of concrete software (based on the metric information), i.e. the formation of a set $\{(sqcxi_1, w_1^{ind}),...,(sqcxi_{(42-nqcxi)}, w_{(42-nqcxi)}^{ind})\} = \{(sqcxi_1, w_1^{ind}),...,(sqcxi_{42}, w_{42}^{ind})\} -$ $(\{(sqcxi_1, w_1^{ind}),...,(sqcxi_{42}, w_{42}^{ind})\} \cap \{sqcxi_1,...,sqcxi_{nqcxi}\})$, where

$\{(sqcxi_1, w_1^{ind}),..., (sqcxi_{(42-nqcxi)}, w_{(42-nqcxi)}^{ind})\} \in SMIW$,

$\{(sqcxi_1, w_1^{ind}),..., (sqcxi_{42}, w_{42}^{ind})\} \in O_{metr}^{w}$, $\{sqcxi_1,..., sqcxi_{nqcxi}\} \in O_{metr}^{real}$ (if the generated set is not empty, then the information in the specification is not sufficient for calculating the software metrics – the more elements in the specified set, the lower the level of sufficiency of information in the requirements);

4) ordering of missing indicators in descending order of values of weights; the numerator of the weighs of each missing quality indicator indicates the number of software metrics that cannot be calculated without this indicator;

5) identification of metrics that cannot be calculated on the basis of available indicators;

6) identification of software characteristics (software project complexity and quality, software complexity and quality) that cannot be calculated on the basis of metrics, which can be calculated using the available indicators

7) making a decision on the need to supplement the specification of software requirements with the necessary indicators in the presence of metrics and characteristics, the value of which cannot be determined on the basis of the indicators available in the specification; the indicators with higher weights must be first considered and included in the specification (the first in the sorted list of missing indicators);

8) forming a request to the developers of specifications for the requirements governing the indicators of quality (with the recommended priority of the supplement); making the necessary additions to the specification with quality information (indicators);

9) repeating steps 1-8 until all indicators are included in the requirements specification and until it is possible to define all metrics and characteristics, or until the conclusion about insufficiency of information on quality in the software requirements specification satisfies the customer.

Method of generating and filling the template of ontology for determining the quality of concrete software (based on the metric information) [74-76, 142-144, 147]. For assessing the sufficiency of quality information (for metric analysis) in the specifications of software requirements by the methods for assessing the sufficiency of quality information (for metric analysis) in the specifications of software requirements based on ontology and

based on weighted ontology, it's necessary to generate and filled a template of ontology for determining the quality of concrete software (based on the metric information). The process of generating and filling the template of ontology for determining the quality of concrete software (based on the metric information) should be automated and simplified to minimize the impact of the human factor and to simplify the assessment of quality information sufficiency (for metric analysis) in software requirements specifications by both developer and customer.

Based on the developed base (universal) ontological model of the subject area "Software Engineering" (part "Specification of software requirements (quality indicators)"), the base ontology of the subject area "Software Engineering" (part "Specification of software requirements (quality indicators)") was developed – Fig. 2.35-2.37. In this ontology, the required quality indicators are presented taking into account the distribution by sections of the specification, due to which the developed ontology is a template of specification of software requirements in terms of indicators and provides visual hints to the user about the location of certain indicators in the software requirements specification.

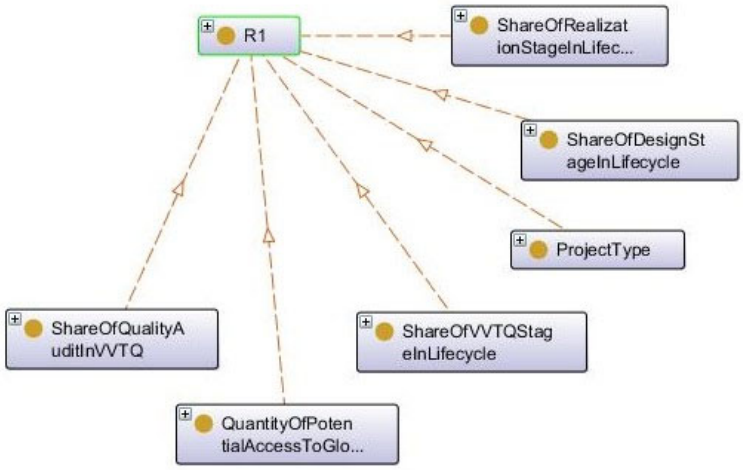

Fig. 2.35. Component of the base ontology of the subject area "Software Engineering" (part "Specification of software requirements (quality indicators)") – indicators from section 1 of the specification (template of section 1 of the specification of requirements in terms of availability of quality indicators)

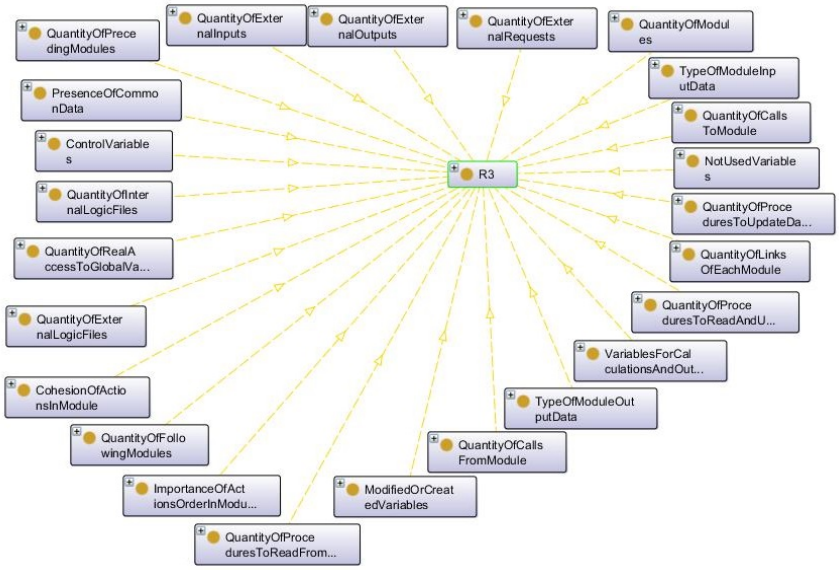

Fig. 2.36. Component of the base ontology of the subject area "Software Engineering" (part "Specification of software requirements (quality indicators)") – indicators form section 3 of the specification (template of section 3 of the specification in terms of availability of quality indicators)

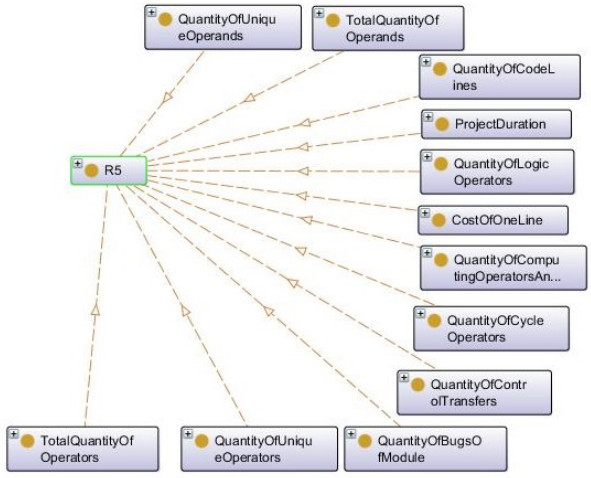

Fig. 2.37. Component of the base ontology of the subject area "Software Engineering" (part "Specification of software requirements (quality indicators)") – indicators form section 5 of the specification (template of section 5 of the specification in terms of availability of quality indicators)

The developed base ontology of the subject area "Software Engineering" (part "Software Quality. Metric analysis"), presented in Fig. 2.27-2.30, contains all 42 software quality indicators that may be contained in the software requirements specification. Therefore, in order to automate and simplify the generation and filling of the template of ontology for determining the quality of concrete software (based on the metric information), we develop a method of generating and filling the template of ontology for determining the quality of concrete software (based on the metric information) based on the base ontology of the subject area "Software Engineering" (part "Software Quality. Metric analysis").

The method of generating and filling the template of ontology for determining the quality of concrete software (based on the metric information) consists of the following steps:

1) to open the base ontology of the subject area "Software Engineering" (part "Software Quality. Metric analysis");

2) to remove from the base ontology all the indicators that were not identified in the specification of the software requirements of the concrete software, using the results of the analysis of the specification of software requirements of the concrete software for the presence of indicators, which are necessary for determining the software metrics, performed in step №1 of the method for assessing the sufficiency of quality information (for metric analysis) in software requirements specifications based on ontology;

3) to save the changes, thus creating an ontology for determining the quality of concrete software.

Method of forming a logical conclusion about the sufficiency of information on quality (for metric analysis) in the specifications of software requirements [74-76, 142-144, 147]. First of all, we develop *production rules for forming a logical conclusion about the sufficiency of quality information (for metric analysis) in the software requirements specifications* (set $PRM = \{ prm_1, .., prm_{43} \}$) based on the base and weighted base ontologies of the subject area Software Engineering (part "Software Quality. Metric analysis"):

1) if the software requirements specification does not have the "Variables For Calculations And Output" indicator (user removed it from the base ontology), then: $chm := chm + 1$ (there is insufficient information for determining the Chepin's metric, so the counter of missing indicators for this metric is increased by 1), i.e. there is insufficient information for determining 1 of the 24 software metrics; `masmt[Variables For Calculations And Output]:=1/42` (in the corresponding element of the array `masmt` the weight of the missing indicator is written, determined by the weighted base ontology of the subject area "Software Engineering (part" Software Quality. Metric analysis")");

2) if the software requirements specification does not have the "Quantity Of Modules" indicator, then: $chm := chm + 1$, $jam := jam + 1$ (counter of missing attributes for Jilb's Absolute metric), $mcm := mcm + 1$ (counter of missing attributes for McClure's metric), $km := km + 1$ (counter of missing attributes for Kafur's metric), $qfbm := qfbm + 1$ (counter of missing attributes for metric "Quantity of found bugs in models"), $eicm := eicm + 1$

(counter of missing attributes for metric “Expected estimate of interfaces complexity”), i.e. there is insufficient information for determining 6 of the 24 metrics; `masmt[Quantity Of Modules]:=6/42`;

3)–42) – rules for the other 40 indicators are similarly formed;

43) if $chm = 0$ and $jam = 0$ and $mcm = 0$ and $km = 0$ and $ch = 0$ and $cpl = 0$ and $gvc = 0$ and $tmm = 0$ and $qfbm = 0$ and $loc = 0$ and $hm = 0$ and $mc = 0$ and $jrm = 0$ and $qps = 0$ and $eicm = 0$ and $sdtt = 0$ and $dst = 0$ and $dc = 0$ and $qac = 0$ and $rp = 0$ and $crc = 0$ and $fpm = 0$ and $eab = 0$ and $dtb = 0$, then in the specification of software requirements, there is sufficient information for determining software complexity and quality based on the metric analysis results, else:

1. there is insufficient quality information (indicators) in the software requirements specification for determining the certain software quality and complexity metrics:

- if $0 < chm \le 5$, then: there is insufficient information for determining Chepin’s metric in the specification of software requirements; if $chm = 5$, then there is no information for determining Chepin’s metric in the software requirements specification;

- if $0 < jam \le 2$, then: there is insufficient information for determining Jilb’s Absolute metric; if $jam = 2$, then there is no information for determining Jilb’s Absolute metric;

- if $0 < mcm \le 5$, then: there is insufficient information for determining McClure’s metric; if $mcm = 5$, then there is no information for determining McClure’s metric;

- if $0 < km \le 4$, then: there is insufficient information for determining Kafur’s metric; if $km = 4$, then there is no information for determining Kafur’s metric;

- if $0 < ch \le 2$, then: there is insufficient information for determining cohesion metric; if $ch = 2$, then there is no information for determining cohesion metric;

- if $0 < cpl \le 5$, then: there is insufficient information for determining coupling metric; if $cpl = 5$, then there is no information for determining coupling metric;

- if $0 < gvc \le 2$, then: there is insufficient information for determining “Global variables calling” metric; if $gvc = 2$, then there is no information for determining “Global variables calling” metric;

- if $0 < tmm \le 3$, then: there is insufficient information for determining “Time of models modification” metric; if $tmm = 3$, then there is no information for determining “Time of models modification” metric;

- if $0 < qfbm \le 2$, then: there is insufficient information for determining “Quantity of found bugs in models” metric; if $qfbm = 2$, then there is no information for determining “Quantity of found bugs in models” metric;

- if $0 < loc \le 1$, then: there is insufficient information for determining “Lines of code” metric; if $loc = 1$, then there is no information for determining “Lines of code” metric;

- if $0 < hm \le 5$, then: there is insufficient information for determining Halstead's metric; if $hm = 5$, then there is no information for determining Halstead's metric;

- if $0 < mc \le 3$, then: there is insufficient information for determining McCabe's metric; if $mc = 3$, then there is no information for determining McCabe's metric;

- if $0 < jrm \le 3$, then: there is insufficient information for determining Jilb's Relative metric; if $jrm = 3$, then there is no information for determining Jilb's Relative metric;

- if $0 < qps \le 2$, then: there is insufficient information for determining "Quantity of program statements" metric; if $qps = 2$, then there is no information for determining "Quantity of program statements" metric;

- if $0 < eicm \le 3$, then: there is insufficient information for determining "Expected estimate of interfaces complexity" metric; if $eicm = 3$, then there is no information for determining "Expected estimate of interfaces complexity" metric;

- if $0 < sdtt \le 2$, then: there is insufficient information for determining "Software design total time" metric; if $sdtt = 2$, then there is no information for determining "Software design total time" metric;

- if $0 < dst \le 3$, then: there is insufficient information for determining "Design stage time" metric; if $dst = 3$, then there is no information for determining "Design stage time" metric;

- if $0 < dc \le 2$, then: there is insufficient information for determining "Design cost" metric; if $dc = 2$, then there is no information for determining "Design cost" metric;

- if $0 < qac \le 4$, then: there is insufficient information for determining "Quality audit cost" metric; if $qac = 4$, then there is no information for determining "Quality audit cost" metric;

- if $0 < rp \le 2$, then: there is insufficient information for determining "Realization productivity" metric; if $rp = 2$, then there is no information for determining "Realization productivity" metric;

- if $0 < crc \le 3$, then: there is insufficient information for determining "Code realization cost" metric; if $crc = 3$, then there is no information for determining "Code realization cost" metric;

- if $0 < fpm \le 5$, then: there is insufficient information for determining "Functional points" metric; if $fpm = 5$, then there is no information for determining "Functional points" metric;

- if $0 < eab \le 2$, then: there is insufficient information for determining "Effort applied by Boehm" metric; if $eab = 2$, then there is no information for determining "Effort applied by Boehm" metric;

- if $0 < dtb \le 2$, then: there is insufficient information for determining "Development time by Boehm" metric; if $dtb = 2$, then there is no information for determining "Development time by Boehm" metric;

2. to sort array `masmt` in descending order of element values (weights of missing indicators);

3. to display the indices of those elements of the sorted array `masmt`, which have a value other than 0 – as the recommended priority of adding indicators to the specification of software requirements.

Based on the production rules for forming a logical conclusion about the sufficiency of quality information (for metric analysis) in the software requirements specifications, we develop *the method of forming a logical conclusion about the sufficiency of information on quality (for metric analysis) in the specifications of software requirements*:

1) according to the Method for assessing the sufficiency of quality information (for metric analysis) in software requirements specifications based on ontology, taking into account the comparative analysis of ontologies, the formation of a set of the indicators $\{ sqcxi_1,..., sqcxi_{(42-nqcxi)} \}$, is performed, which are missing in the ontology for determining the quality of concrete software (based on the metric information);

2) according to the method of searching in the width in the forward direction, in the subset of production rules $\{ prm_1,.., prm_{42} \}$ there is a search for a rule for each element of the set $\{ sqcxi_1,..., sqcxi_{(42-nqcxi)} \}$, according to which the counters of missing indicators for metrics are calculated;

3) according to the rule prm_{43}, the analysis of information on quality (indicators) in the specification for sufficiency is performed, and, in case of insufficiency, conclusions are formed, for determining of which software metrics the information is insufficient, as well as the sorted (by weights) list of missing indicators is formed as a recommended priority of adding indicators to the software requirements specification;

4) the numerical assessment of the sufficiency of the volume of quality information (indicators), available in the requirements, is performed – should strive for this assessment to be as large as possible, i.e. to go to 1:

$$q_{metr} = \frac{chm}{5} + \frac{jam}{2} + \frac{mcm}{5} + \frac{km}{4} + \frac{ch}{2} + \frac{cpl}{5} + \frac{gvc}{2} + \frac{tmm}{3} + \frac{qfbm}{2} + \frac{loc}{1} + \frac{hm}{5} + \\ + \frac{mc}{3} + \frac{jrm}{3} + \frac{qps}{2} + \frac{eicm}{3} + \frac{sdtt}{2} + \frac{dst}{3} + \frac{dc}{2} + \frac{qac}{4} + \frac{rp}{2} + \frac{crc}{3} + \frac{fpm}{5} + \frac{eab}{2} + \frac{dtb}{2}, \quad (2.32)$$

where q_{metr} – the number of metrics that cannot be calculated based on the indicators, available in the specification; numbers in fraction numerators indicate the number of indicators missing in the specification for a certain metric, numbers in fraction denominators indicate the number of required indicators for each metric;

$$D_{metr} = \frac{24 - q_{metr}}{24}, \quad (2.33)$$

where D_{metr} – numerical assessment of the sufficiency of the amount of information (indicators) available in the specification;

5) numerical assessment of the sufficiency of the available (after supplementing) quality information (indicators) in the software requirements specification (taking into account that the methods for assessing the sufficiency of quality information (for metric analysis) in the software requirements specifications based on ontology and based on weighted ontology are iterative, after forming a conclusion about the insufficient information in the specification, the specification of requirements will be supplemented with the necessary indicators, which will increase the sufficiency of information on quality in the requirements):

$$q'_{metr} = \frac{chm'}{5} + \frac{jam'}{2} + \frac{mcm'}{5} + \frac{km'}{4} + \frac{ch'}{2} + \frac{cpl'}{5} + \frac{gvc'}{2} + \frac{tmm'}{3} + \\ + \frac{qfbm'}{2} + \frac{loc'}{1} + \frac{hm'}{5} + \frac{mc'}{3} + \frac{jrm'}{3} + \frac{qps'}{2} + \frac{eicm'}{3} + \frac{sdtt'}{2} + \frac{dst'}{3} + , \\ + \frac{dc'}{2} + \frac{qac'}{4} + \frac{rp'}{2} + \frac{crc'}{3} + \frac{fpm'}{5} + \frac{eab'}{2} + \frac{dtb'}{2} \quad (2.34)$$

where q'_{metr} – the number of metrics that cannot be calculated based on the indicators, available in the specification after specification's supplementing; numbers in fraction numerators indicate the number of indicators missing in the specification for a certain metric (after supplementing);

$$D'_{metr} = \frac{24 - q'_{metr}}{24}, \quad (2.35)$$

where D'_{metr} – numerical assessment of the sufficiency of the amount of information (indicators) available in the specification (after supplementing);

6) the gain of the sufficiency of the amount of information on quality (indicators) in the specification of software requirements (after supplementing the specification with the necessary indicators) is calculated:

$$\Delta q_{metr} = q_{metr} - q'_{metr} = (\frac{chm}{5} - \frac{chm'}{5}) + (\frac{jam}{2} - \frac{jam'}{2}) + ... + (\frac{dtb}{2} - \frac{dtb'}{2}), \quad (2.36)$$

where Δq_{metr} – the number of metrics that became possible to calculate after the supplementing the specification with the indicators;

$$\Delta D_{metr} = D'_{metr} - D_{metr} = \frac{\Delta q_{metr}}{24}, \quad (2.37)$$

where ΔD_{metr} – the gain of the sufficiency of the amount of information on quality (indicators) in the specification (after supplementing).

2.4. Intelligent Information Technology for Assessing the Sufficiency of Quality Information in the Specifications of the Software Requirements

Based on the modeling of information flows in the formation of the specification of software requirements and the study of information flows in the process of assessing the sufficiency of information on quality in the specifications of software requirements, as well as taking into account the theoretical principles of information technology of assessing the sufficiency of information on quality in the specifications of software requirements, the information technology for assessing the sufficiency of quality information in the specifications of the software requirements is developed [75, 144] (as a set of processes that uses tools and methods of accumulation, processing and transmission of primary information for obtaining new quality information about the state of an object, process or phenomenon), structural scheme of which is presented in Fig. 2.38.

The purpose of the developed information technology is to assess the sufficiency of information on quality in the specification of requirements for the developed software.

The basis of information technology for assessing the sufficiency of quality information in the specifications of the software requirements are: methods of generating and filling the template of ontology for determining the quality of concrete software based on ISO 25010 and metric information; methods for assessing the sufficiency of quality information (according to ISO 25010 and for metric analysis) in software requirements specifications based on ontologies and based on weighted ontologies; methods of forming a logical conclusion about the sufficiency of information on quality (according to ISO 25010 and for metric analysis) in the specifications of software requirements; subsystem for assessing the sufficiency of information on quality in the software requirements specifications based on comparative analysis of ontologies, which is based on the above methods.

Information technology for assessing the sufficiency of quality information in the specifications of the software requirements provides:

- support for the software quality assessment process in the early stages of the life cycle based on the assessment of the sufficiency of quality information in the software requirements specifications;
- processing of quality information in the software requirements specifications by software agents (bots), without the participation of specialists, that provides the ability to automate such processes and eliminate the subjective influence of specialists, as well as the safety of this information in the software company in case of dismissal of specialists;
- conclusion on the sufficiency of quality information in the software requirements specifications;
- the priority of supplementing the requirements specification with the necessary information (in case of insufficient information) by forming a request;
- numerical assessment of the sufficiency of the amount of quality information available in the requirements specification;

- numerical assessment of the gain of the sufficiency of the amount of information on quality in the specification of requirements after its supplementing.

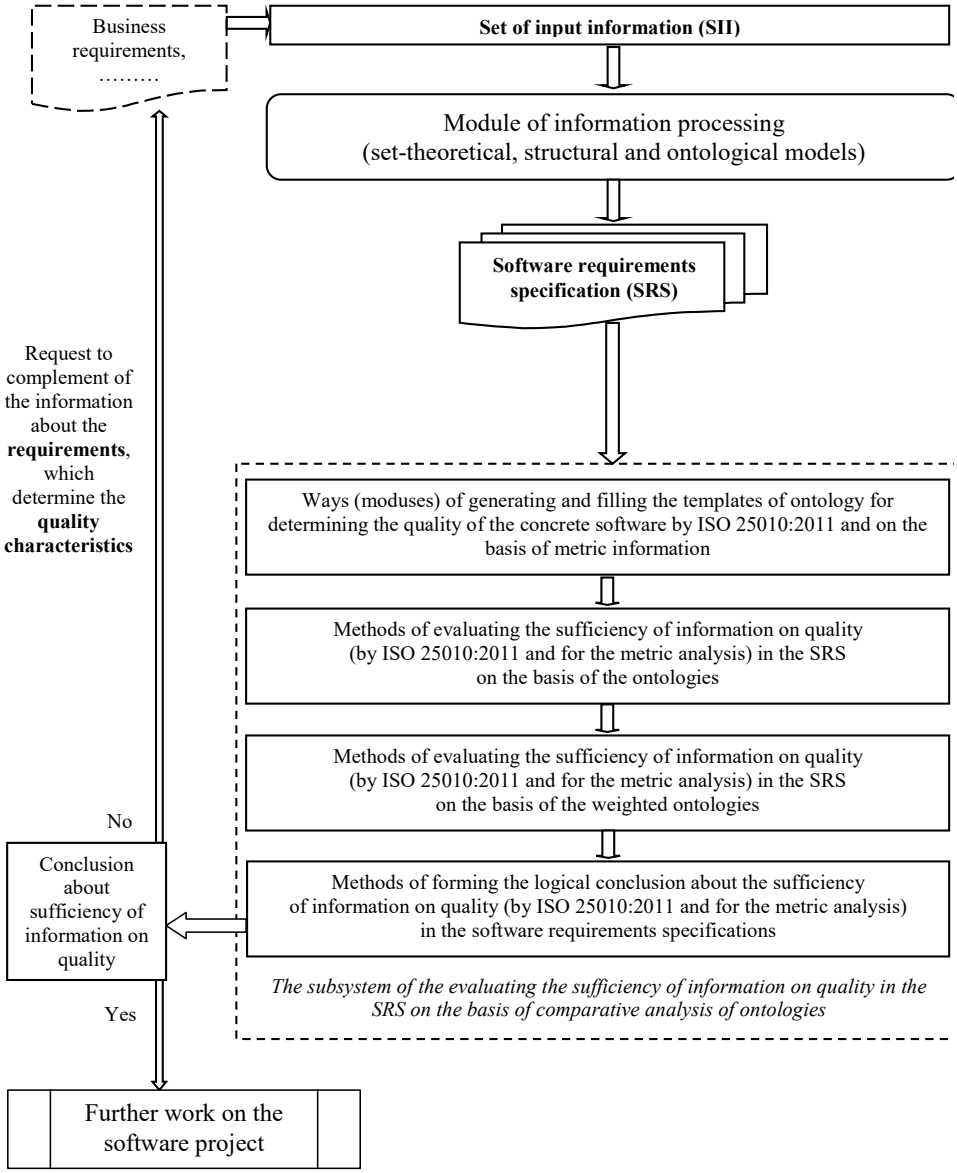

Fig. 2.38. Structural scheme of information technology for assessing the sufficiency of quality information in the specifications of the software requirements

With the purpose of the developing the applied principles of information technology for assessing the sufficiency of quality information in the software requirements specifications, it is necessary to design and implement a subsystem for assessing the sufficiency of quality information in the software requirements specifications based on comparative analysis of ontologies.

When implementing this subsystem, the processes of assessing the sufficiency of information on quality in the software requirements specifications should be maximally automated in order to minimize the impact of the human factor and to simplify the implementation of this assessment by both the developer and the customer.

Subsystem for assessing the sufficiency of quality information in the software requirements specifications based on comparative analysis of ontologies [75, 144]. On the input of the subsystem for assessing the sufficiency of information on quality in the software requirements specifications based on comparative analysis of ontologies are given sets: 1) $\{qms_1,...qms_{nm}\}$ ($nm \le 138$)of available in the software requirements specification of the above quality attributes, which are necessary for assessing the software quality characteristics and sub-characteristics; 2) $\{sqcxi_1,...,sqcxi_{nqcxi}\}$ ($nqcxi \le 42$) of available in the software requirements specification of the above indicators required to perform the metric analysis.

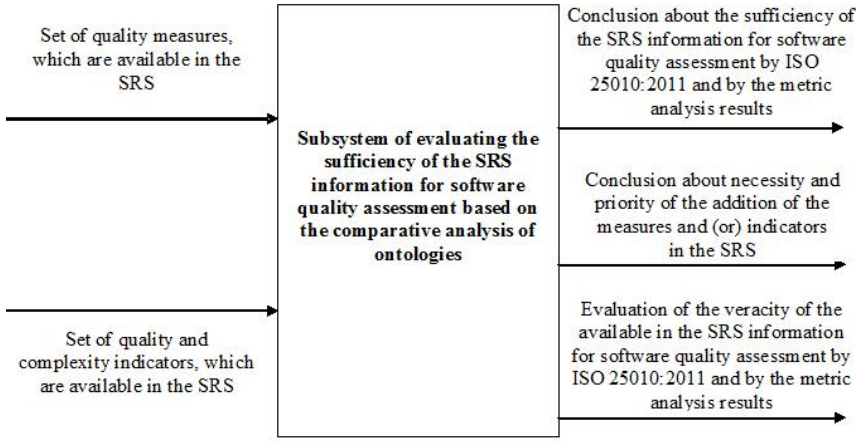

Fig. 2.39. The conceptual scheme of the subsystem for assessing the sufficiency of information on quality in the specifications of the software requirements based on the comparative analysis of ontologies [144]

The results of such a subsystem are:

1) conclusion on the sufficiency of information on quality (according to ISO 25010) in the specification of requirements for concrete software;

2) recommendations on the need and priority of supplementing the specification of

software requirements with quality attributes (in case of insufficiency of attributes);

3) assessment of the sufficiency of the amount of quality information (attributes) available in the specification of requirements;

4) conclusion on the sufficiency of quality information (for metric analysis) in the specification of requirements for concrete software;

5) recommendations on the need and priority of supplementing the specification of software requirements with indicators (in case of insufficiency of indicators);

6) assessment of the sufficiency of the amount of information (indicators) available in the specification of requirements.

The conceptual scheme of the subsystem for assessing the sufficiency of information on quality in the specifications of the software requirements based on the comparative analysis of ontologies is presented in Fig. 2.39 [144].

The structure of the subsystem for assessing the sufficiency of information on quality in the specifications of the software requirements based on the comparative analysis of ontologies is presented in Fig. 2.40 [144].

The subsystem for assessing the sufficiency of information on quality in the specifications of the software requirements based on the comparative analysis of ontologies consists of the following components:

1) the module of introduction of the attributes of the specification of software requirements is an integral part of the user interface; reads user information on available attribute values $\{qms_1,...qms_{nm}\}$ ($nm \le 138$)in section 3 of the software requirements specification; the user is offered a list of all 138 quality attributes in the base ontology, from which he removes attributes that are not in the analyzed specification, forming an ontology for determining the quality of the concrete software;

2) the module of the introduction of indicators of the specification of requirements to the software is a component of the user interface; reads user information on the available values of indicators $\{sqcxi_1,...,sqcxi_{nqcxi}\}$ ($nqcxi \le 42$)in sections 1, 3, 5 of the specification of software requirements; the user is offered a list of all 42 indicators in the base ontology, from which he deletes indicators that are not in the analyzed specification, forming an ontology for determining the quality of the concrete software (based on metric information);

3) module of the user support is an integral part of the user interface; provides the user with information about: the structure of the specification of software requirements (in the form of ontologies (Fig. 2.24, 2.35-2.37), which are templates of the specification of software requirements in terms of quality attributes and indicators); attributes required for software quality assessment according to ISO 25010; indicators needed to assess the quality of software based on the results of metric analysis; the process of forming the results of the subsystem;

4) module for assessing the sufficiency of information in the specification of software requirements for determining the quality of software according to ISO 25010 – generating and filling the template of ontology for determining the quality of the concrete software are performed, taking into account the attributes $\{qms_1,...qms_{nm}\}$ ($nm \le 138$) entered in the

module of the introduction of the attributes of the specification of software requirements, according to the method of generating and filling the ontology template for determining the quality of the concrete software. The comparative analysis of the ontologies for determining the quality of the concrete software with the base ontology of the subject area "Software Engineering" (part "Software Quality") is performed, the result of which is a list of attributes missing in the specification. According to the method of forming a logical conclusion about the sufficiency of information on quality (according to ISO 25010:2011) in the specifications of the software requirements, the results of comparative analysis of ontology for the concrete software with the base ontology are processed. If the comparative analysis of ontologies does not reveal discrepancies (the list of missing attributes is empty), then the information on the quality of the specification is sufficient for determining the quality of software according to ISO 25010, then the module of results display gives the user of the subsystem the conclusion "Information of the considered specification is sufficient for determining the software quality according to the ISO 25010 standard". If the comparative analysis of ontologies reveals discrepancies, the specification does not have insufficient attributes for determining the certain sub-characteristics and characteristics of software quality, then the module of results display gives the user of the subsystem the conclusion "Information of the considered specification is insufficient for determining the software quality according to the ISO 25010 standard", followed by a comparative analysis of the ontology for determining the quality of concrete software with a weighted base ontology of the subject area "Software Engineering" (part "Software Quality"). Next, the production rule is searched (in the production rules for forming a logical conclusion about the sufficiency of information on quality (according to ISO 25010:2011) in the software requirements specifications contained in the knowledge base) for each missing attribute in the specification, according to which the subsystem outputs conclusion to the user on the sub-characteristics and quality characteristics of the software, for determination of which in the specification is insufficient information, as well as sorting all missing attributes of the specification in descending order of weights, i.e. the recommended priority of their supplementing the software requirements. With the help of the module of the results display, the generated recommendations are issued to the user of the subsystem in the form of conclusion "For increasing the sufficiency of the volume of quality in the software requirements specification, it's recommended to supplement the specification information with the attributes in the following sequence: ", followed by a list of all missing attributes in accordance with the established priority of supplementing them to the specification. In addition, the subsystem, based on the results of certain production rules, according to step 4 of the method of forming a logical conclusion about the sufficiency of quality information (according to ISO 25010:2011) in the software requirements specifications, performs a quantitative assessment of the sufficiency of the requirements information (attributes) for assessing the quality of software by sub-characteristics and characteristics and displays conclusion to the user in the form: "Sufficiency of the available information in the specification of information (attributes) for assessing the quality of software according to ISO 25010 is: ", followed by quantitative assessment of the sufficiency;

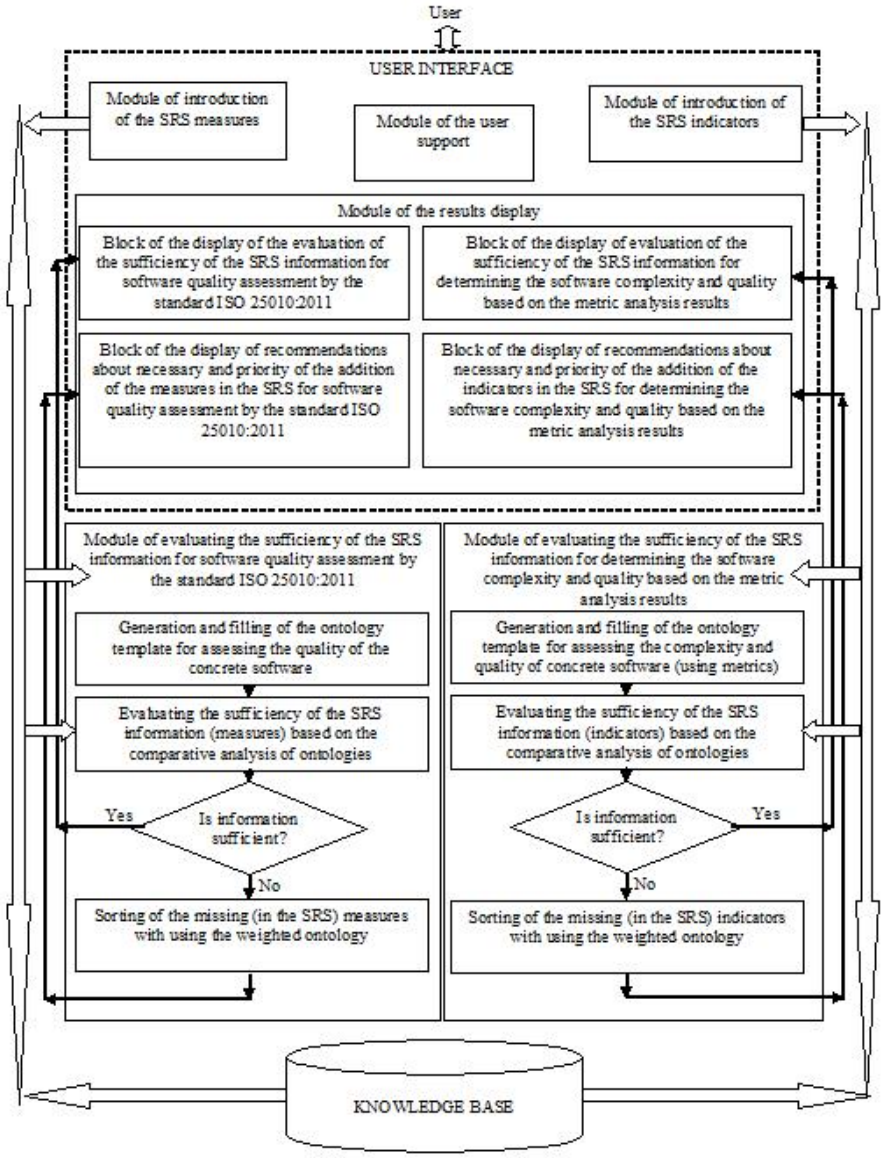

Fig. 2.40. The structure of the subsystem for assessing the sufficiency of information on quality in the specifications of the software requirements based on the comparative analysis of ontologies [144]

5) module for assessing the sufficiency of information in the specification of software requirements for determining the quality of software based on the metric analysis – works similarly to the module for assessing the sufficiency of information in the specification of

software requirements for determining the quality of software according to ISO 25010, i.e. first of all generating and filling the template of ontology for determining the quality of the concrete software (based on the metric information) are performed, taking into account the indicators $\{ sqcxi_1,...,sqcxi_{nqcxi} \}$ ($nqcxi \le 42$) entered in the module of the introduction of the indicators of the specification of software requirements, according to the method of generating and filling the ontology template for determining the quality of the concrete software (based on the metric information). The comparative analysis of the ontologies for determining the quality of the concrete software (based on the metric information) with the base ontology of the subject area "Software Engineering" (part "Software Quality. Metric Analysis") is performed, the result of which is a list of indicators missing in the specification. According to the method of forming a logical conclusion about the sufficiency of information on quality (for metric analysis) in the specifications of the software requirements, the results of comparative analysis of ontology for the concrete software (based on the metric information) with the base ontology are processed. If the comparative analysis of ontologies does not reveal discrepancies (the list of missing indicators is empty), then the information on the quality of the specification is sufficient for determining the quality of software based on the metric analysis results, then the module of results display gives the user of the subsystem the conclusion "Information of the considered specification is sufficient for determining the software quality based on the metric analysis results". If the comparative analysis of ontologies reveals discrepancies, the specification does not have insufficient indicators for determining the certain software metrics, then the module of results display gives the user of the subsystem the conclusion "Information of the considered specification is insufficient for determining the software quality based on the metric analysis results", followed by a comparative analysis of the ontology for determining the quality of concrete software (based on the metric information) with a weighted base ontology of the subject area "Software Engineering" (part" Software Quality. Metric Analysis"). Next, the production rule is searched (in the production rules for forming a logical conclusion about the sufficiency of information on quality (for metric analysis) in the software requirements specifications contained in the knowledge base) for each missing indicator in the specification, according to which the subsystem outputs conclusion to the user on the metrics of the software, for determination of which in the specification is insufficient information, as well as sorting all missing indicators of the specification in descending order of weights, i.e. the recommended priority of their supplementing the software requirements. With the help of the module of the results display, the generated recommendations are issued to the user of the subsystem in the form of conclusion "For increasing the sufficiency of the volume of quality in the software requirements specification, it's recommended to supplement the specification information with the indicators in the following sequence: ", followed by a list of all missing indicators in accordance with the established priority of supplementing them to the specification. In addition, the subsystem, based on the results of certain production rules, according to step 4 of the method of forming a logical conclusion about the sufficiency of quality information (for metric analysis) in the software requirements specifications, performs a quantitative assessment of the sufficiency of the requirements information (indicators) for assessing the quality of software by

metrics and displays conclusion to the user in the form: "Sufficiency of the available information in the specification of information (indicators) for assessing the quality of software based on the metric analysis results is: ", followed by quantitative assessment of the sufficiency;

6) knowledge base – contains a base and weighted base ontology of the subject area "Software Engineering" (parts "Software Quality", "Software Quality. Metric Analysis"), the base ontology of the subject area "Software Engineering" (parts "Specification of software requirements (quality attributes)", "Specification of software requirements (quality indicators)", generated by users of the ontology for determining the quality of the concrete software and ontology for determining the quality of the concrete software (based on metric information), as well as production rules for forming a logical conclusion about the sufficiency of quality information (according to ISO 25010:2011 and for metric analysis) in the software requirements specifications;

7) module of the results display – is an integral part of the user interface; it's used for displaying the conclusions of subsystem about sufficiency/insufficiency of information to the user.

Advantages of information technology for assessing the sufficiency of information on quality in the software requirements specifications [75]. The modern development of information technologies requires decisive changes due to the emergence of powerful technical resources for processing large amounts of information, as well as due to the intellectualization of information processing, which allows solving new classes of problems. The described needs are prerequisites for the development of theoretical and applied foundations of new-generation information technologies. In this case, the content and methods of information processing are significantly influenced by the characteristics of the subject areas for which information technology is developed. Today, the software engineering industry needs special attention to the development and implementation of new-generation information technologies in terms of software quality assurance.

The problem of software quality assurance today remains one of the main problems of software engineering. Early stages of the software life cycle, in particular, the stage of formation and formulation of requirements, are the least formalized and most expensive, because the cost of correcting the incorrect requirements identified after product release is hundreds of times higher than the cost of correction of the specification problems, which are identified at the stage of requirements formulation. The risks of the insufficiently worked-out stage of formation and formulation of requirements are non-compliance with project deadlines and financial overspending, which can lead to the closure of the project, and even the collapse of the software company due to its financial instability.

When formulating requirements, it is important to adhere to the requirements of the standards governing the software development process. However, most of these standards are presented in natural language in text form, so there is no mechanism to verify the results of the implementation of the standard in the software development process. When developing software at the junction of subject areas, the task is significantly complicated, because it is necessary to implement the standards of the subject area for which the software is developed.

Now information technology involves human-machine interaction at all stages of information processing (Fig. 2.41), during which all information is interpreted by a person, which often leads to loss of essential information. But the need to process large amounts of information and the development of various areas of intellectualization of information and data are prerequisites for the transition to a new level of information processing through the development of new-generation information technologies in which people are eliminated from information processing and knowledge acquisition (Fig. 2.42).

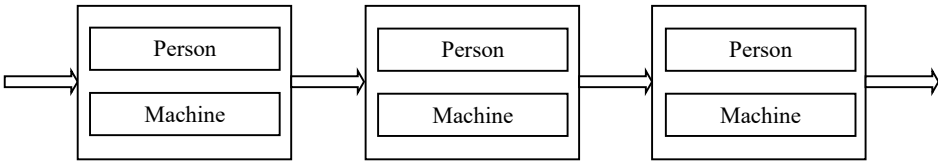

Fig. 2.41. Modern information technologies: human-machine interaction is provided at all stages of information processing

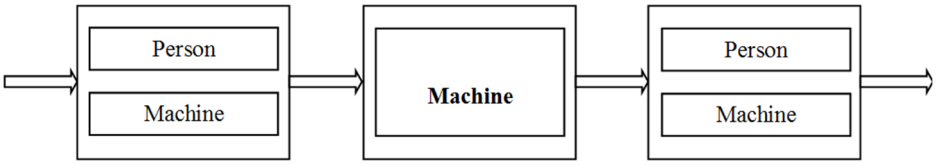

Fig. 2.42. Information technology for assessing the sufficiency of information on quality in the specifications of software requirements: partial elimination of a person from the processes of information processing and knowledge acquisition

The basis of information technology for assessing the sufficiency of quality information in the specifications of software requirements is the use of ontologies – base ontologies of the subject area "Software Engineering" (parts "Software Quality", "Software Quality. Metric Analysis"), ontologies for determining the quality of the concrete software (according to ISO 25010:2011 and based on metric information) for providing the quality information in the specification of software requirements. Therefore, information technology, developed on the basis of the concept of the Semantic Web, provides duplication and gaps in this information based on the visualization of missing logical connections, as well as the ability to process quality information by intelligent and non-intelligent virtual agents (bots), without human intervention, that provides the ability to automate such processes and eliminate the subjective influence of man, as well as the safety of this information in the software company in case of dismissal of a specialist.

The development of information technology for assessing the sufficiency of information on quality in the specifications of software requirements is possible through the

development of software ontology-based agents (bots). The introduction of advanced information technology in software companies automates the process of verifying the sufficiency of information on quality in the specifications of software requirements, will allow the use of software agents to analyze and supplement the specification, to assess the initial stages of the life cycle, to minimize information and financial losses.

2.5. Results of Functioning the Intelligent Information Technology for Assessing the Sufficiency of Quality Information in the Specifications of the Software Requirements

Assessment of the sufficiency of quality information (according to ISO 25010:2011) in the software requirements specifications [74-76, 142-144]. During the experiment, *the specification of requirements for a software agent for improving the security of computer systems' software* was analyzed. The software quality attributes required for determining the software quality sub-characteristics and characteristics available in a particular specification have been identified. Based on the method of generating and filling the template of ontology for determining the quality of concrete software, an ontology for determining the quality of the concrete software is developed, which is formed by components for determining: Functional Suitability (Fig. 2.43), Compatibility (Fig. 2.44), Performance Efficiency (Fig. 2.45), Portability (Fig. 2.46), Usability (Fig. 2.47), Reliability (Fig. 2.48), Security (Fig. 2.49), Maintainability (Fig. 2.50) of the concrete software.

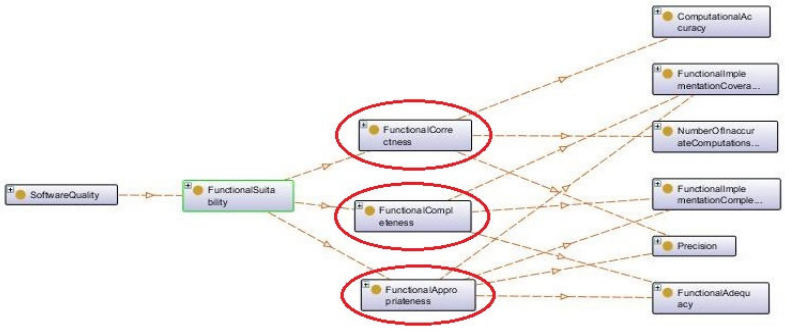

Fig. 2.43. Component of ontology for Functional Suitability for a software agent for improving the security of computer systems' software

Comparison of the developed ontology for the software agent for improving the security of computer systems' software (Fig. 2.43-2.50) with fragments of the base ontology for the subject area "Software Engineering", part "Software Quality" (Fig. 2.6-2.13) made it

possible to find that in the developed ontology of the concrete software project there are no 4 attributes (Fig. 2.51): "Number Of Functions", "Operation Time", "Number Of Data Items", "Number Of Test Cases", then the set of missing attributes $\{qms_1^{AS},...,qms_4^{AS}\} = \left\{ \begin{matrix} "NumberOfFunctions","OperationTime", \\ "NumberOfDataItems","NumberOfTestCases" \end{matrix} \right\}$.

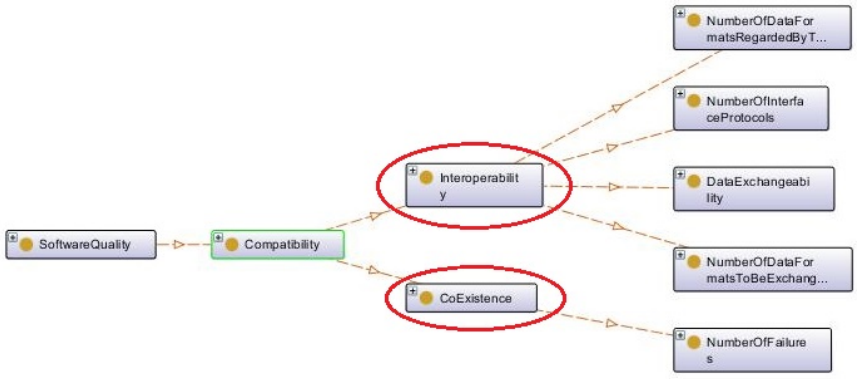

Fig. 2.44. Component of ontology for Compatibility for a software agent for improving the security of computer systems' software

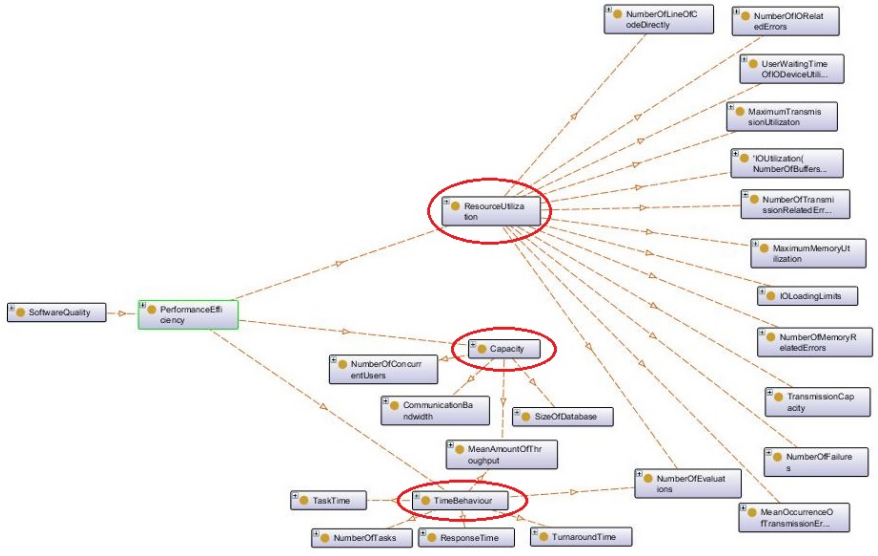

Fig. 2.45. Component of ontology for Performance Efficiency for a software agent for improving the security of computer systems' software

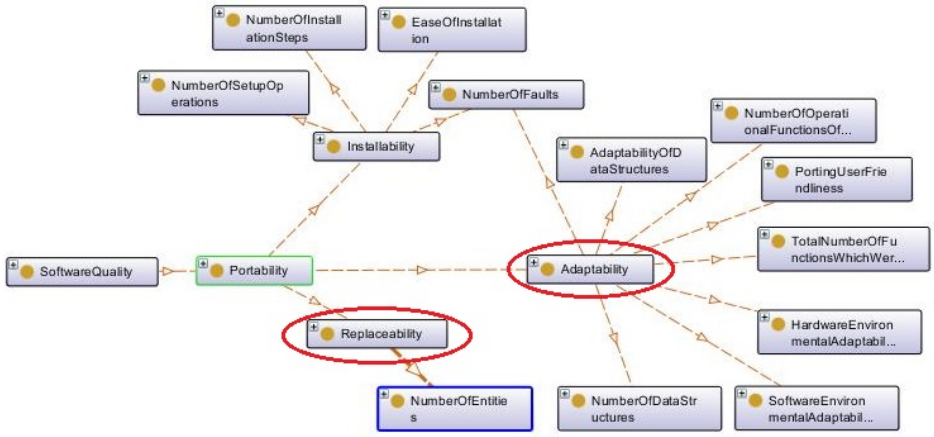

Fig. 2.46. Component of ontology for Portability for a software agent for improving the security of computer systems' software

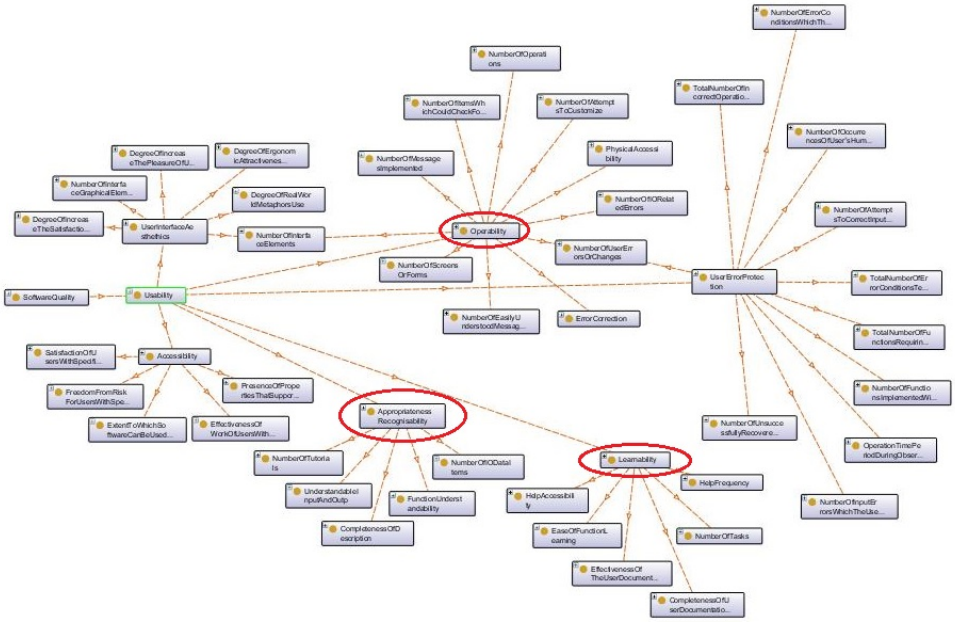

Fig. 2.47. Component of ontology for Usability for a software agent for improving the security of computer systems' software

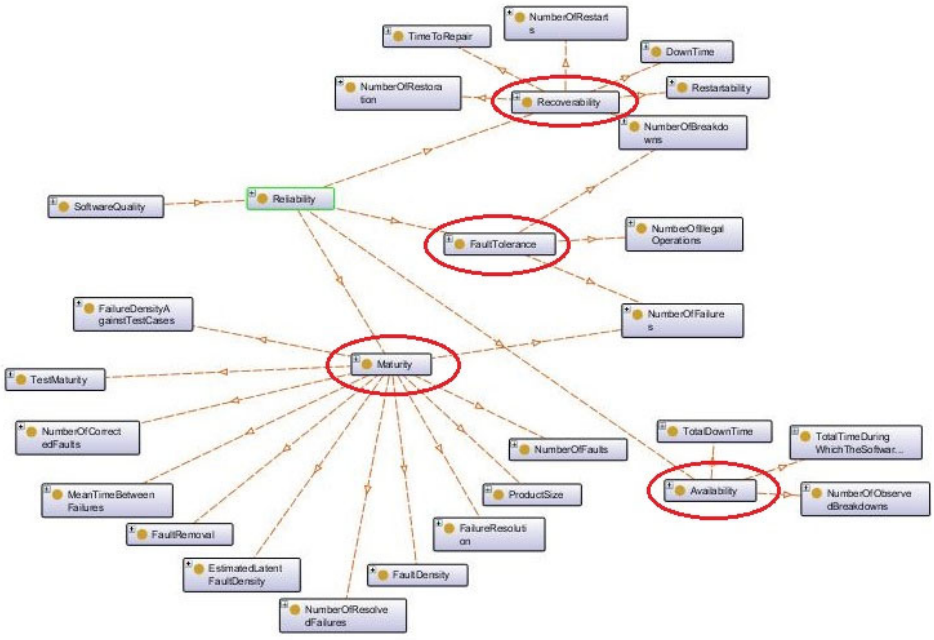

Fig. 2.48. Component of ontology for Reliability for a software agent for improving the security of computer systems' software

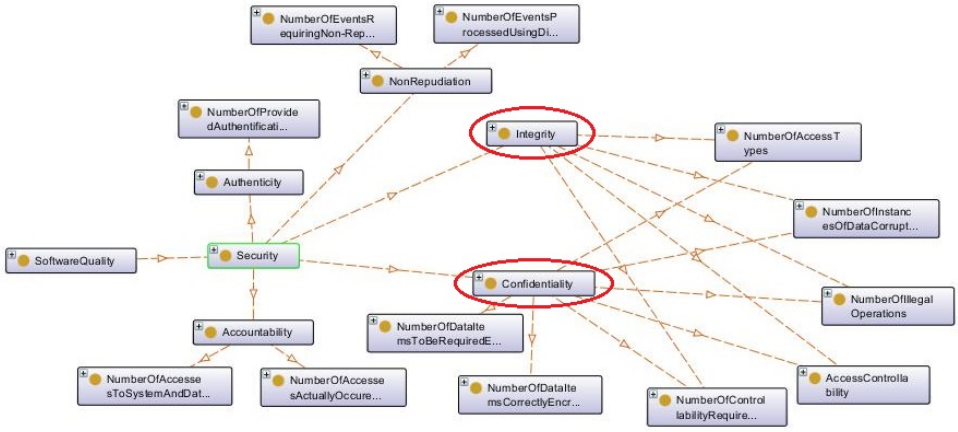

Fig. 2.49. Component of ontology for Security for a software agent for improving the security of computer systems' software

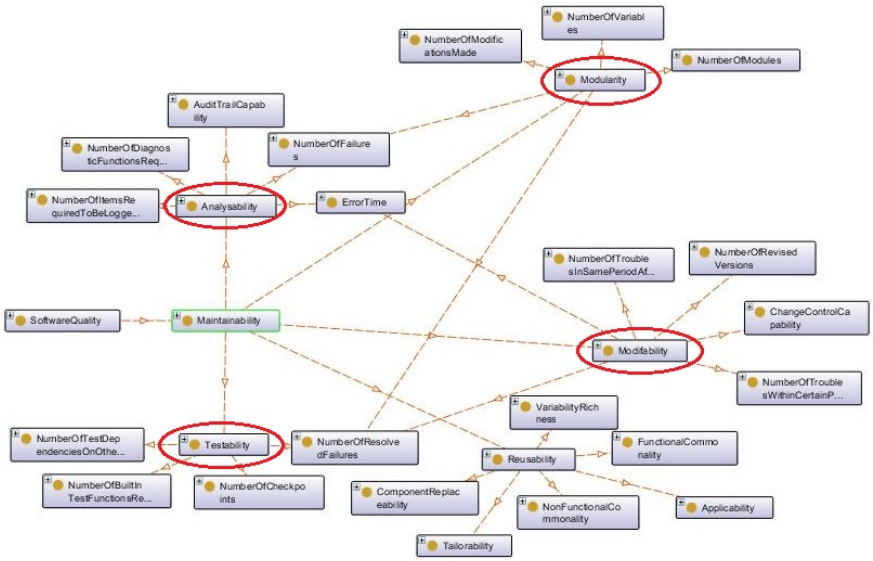

Fig. 2.50. Component of ontology for Maintainability for a software agent for improving the security of computer systems' software

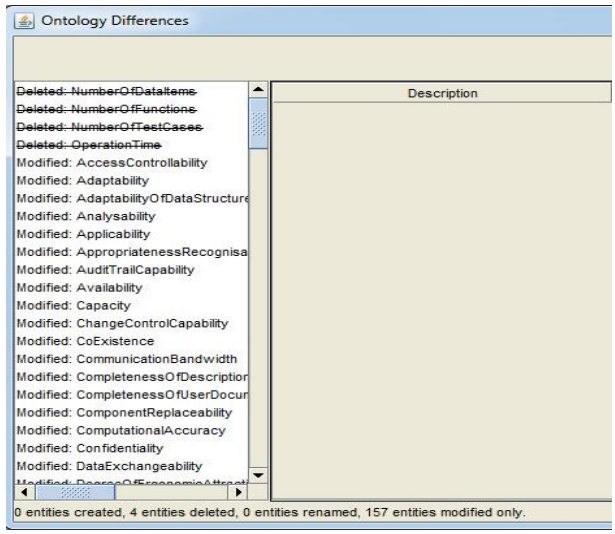

Fig. 2.51. Comparison of the ontology for determining the quality of concrete software (software agent for improving the security of computer systems' software) with the base ontology of the subject area "Software Engineering" (part "Software Quality")

According to the method of forming a logical conclusion about the sufficiency of information on quality (according to ISO 25010:2011) in the software requirements specifications, a search of the production rule is made for each element of the set $\{qms_1^{AS},...,qms_4^{AS}\}$, according to which the attributes, which are missing in the specification for determining the software quality sub-characteristics and characteristics, are calculated, – the results of such a search are shown in Table 2.1.

Tab. 2.1. List of software quality attributes missing in the specification and the results of the method of forming a logical conclusion about the sufficiency of information on quality (according to ISO 25010:2011) in the software requirements specifications

Missing attribute	Number of the rule	Results of the method (rule)
Number of Functions	1	$fc=1$, $fa=1$, $ft=1$, $ar=1$, $lb=1$, $ob=1$, $md=1$, $mfb=1$, $cex=1$, $ab=1$, $rb=1$; $fy=2$, $ry=1$, $uy=3$, $my=2$, $cy=1$, $py=2$; mas[Number Of Functions]=11/138
Operation Time	2	$fcr=1$, $fa=2$, $ma=1$, $av=1$, $rvb=1$, $tb=1$, $ru=1$, $lb=2$, $ob=2$, $md=2$, $mfb=2$, $tst=1$, $cf=1$, $ig=1$, $cex=2$, $io=1$, $ab=2$; $fy=4$, $ry=4$, $ey=2$, $uy=5$, $my=5$, $sy=2$, $cy=3$, $py=3$; mas[Operation Time]=17/138
Number of Data Items	7	$fcr=2$, $ccy=1$, $anb=1$, $cf=2$, $ig=2$, $cex=3$, $ab=3$, $rb=2$; $fy=5$, $ey=3$, $my=6$, $sy=4$, $cy=4$, $py=5$; mas[Number Of Data Items] =8/138
Number of Test Cases	13	$ma=2$, $ft=2$, $tst=2$, $cf=3$, $ig=3$; $ry=6$, $my=7$, $sy=6$; mas[Number Of Test Cases]=5/138

According to rule №139, it was found that the specification of the requirements for the software agent to improve the security of computer systems' software has insufficient attributes for determining certain sub-characteristics of software quality, namely for sub-characteristics: Functional Completeness, Functional Correctness, Functional

Appropriateness, Maturity, Availability, Fault Tolerance, Recoverability, Time Behaviour, Resource Utilization, Capacity, Appropriateness Recognisability, Learnability, Operability, Modularity, Analysability, Modifability, Testability, Confidentiality, Integrity, CoExistence, Interoperability, Adaptability, Replaceability. Sub-characteristics, for the determination of which there are insufficient attributes in the specification, are circled in Fig. 2.43-2.50.

According to Part 1 of Rule No. 140, it was found that the specification of requirements of the software agent for improving the security of computer systems' software has insufficient attributes for all eight software quality characteristics.

Thus, the absence of four attributes in the software requirements led to the impossibility of calculating 23 (from 31) software quality sub-characteristics, to the impossibility of calculating all software quality characteristics with a high level of veracity and, accordingly, to the impossibility of determining the future software quality with a high level of veracity. Then the subsystem for assessing the sufficiency of quality information in the software requirements specifications based on a comparative analysis of ontologies of information technology for assessing the sufficiency of quality information in the software requirements specifications concluded: "The information in this specification is insufficient for determining the quality of software according to ISO 25010."

After establishing the fact of insufficiency of information on quality (attributes) in the specification of requirements to the software agent for improving the security of computer systems' software, according to 2-nd and 3-rd parts of rule No.140, sorting of array `mas` on descending values of elements (weights of missing attributes) is performed and the indices of those elements of the sorted array `mas` are displayed, which have a value other than 0 – Table 2.2.

Tab. 2.2. List of software quality attributes missing in the specification in descending order of weights

Missing quality attributes (indexes of elements of array `mas`)	**Weights (values of elements of array `mas`)**
Operation Time	17/138
Number of Functions	11/138
Number of Data Items	8/138
Number of Test Cases	5/138

Table 2.2 shows the priority and order of consideration and supplementing the quality attributes to the specification of requirements of a software agent for improving the security of computer systems' software. Therefore, the subsystem for assessing the sufficiency of quality information in the software requirements specifications based on a comparative analysis of ontology of information technology for assessing the sufficiency of quality information in the software requirements specifications concluded: “For increasing the sufficiency of quality information in the specification, it is recommended to supplement the

attributes in the following sequence: 1) Operation Time; 2) Number Of Functions; 3) Number Of Data Items; 4) Number Of Test Cases".

Next, the sufficiency of the amount of information available in the requirements specification for assessing the quality of the software was assessed:

- by sub-characteristics:

$$q_{schr}^{AS} = \frac{1}{4}+\frac{2}{5}+\frac{2}{6}+\frac{2}{14}+\frac{1}{4}+\frac{2}{5}+\frac{1}{7}+\frac{1}{7}+\frac{1}{14}+\frac{1}{5}+\frac{1}{6}+\frac{2}{8}+\frac{2}{13}+\frac{0}{11}+$$
$$+\frac{0}{6}+\frac{0}{5}+\frac{2}{7}+\frac{0}{6}+\frac{1}{6}+\frac{2}{8}+\frac{2}{6}+\frac{3}{10}+\frac{3}{8}+\frac{0}{2}+\frac{0}{2}+\frac{0}{1}+\frac{3}{4}+\frac{1}{5}+\frac{3}{11}+\frac{0}{4}+\frac{2}{3}=6{,}5,$$

$$D_{schr}^{AS} = \frac{31-6{,}5}{31} = 0{,}79;$$

- by characteristics:

$$q_{chr}^{AS} = \frac{5}{15}+\frac{6}{30}+\frac{3}{26}+\frac{5}{49}+\frac{7}{33}+\frac{6}{23}+\frac{4}{9}+\frac{5}{18}=1{,}95,$$

$$D_{chr}^{AS} = \frac{8-1{,}95}{8} = 0{,}76.$$

Thus, the subsystem for assessing the sufficiency of information on quality in the specifications of software requirements based on a comparative analysis of ontologies of information technology for assessing the sufficiency of information on quality in the specifications of software requirements displays the conclusion to the user: "Sufficiency of the volume of information is 79% for software quality evaluation by sub-characteristics and 76% for software quality evaluation by characteristics".

Because the proposed methods for assessing the sufficiency of quality information (according to ISO 25010:2011) in the software requirements specifications based on ontology and weighted ontology are iterative, and for this software project there are sub-characteristics and characteristics for determination of which there are insufficient attributes in the specification, a request was made to supplement the software requirements specification. After supplementing the requirements specification, an ontology (version 2) for determining the quality of concrete software was re-developed. A comparative analysis of the developed ontology (version 2) with fragments of the base ontology of the subject area "Software Engineering" (part "Software Quality") found that the specification of software requirements was supplemented by attributes "Number Of Functions" (2-nd in the sorted list), "Number of Data Items" (3-rd in the sorted list), then the set of missing attributes $\{ qms_1'^{AS}, qms_2'^{AS} \} = \{"OperationTime","NumberOfTestCases"\}$.

According to the method of forming a logical conclusion about the sufficiency of information on quality (according to ISO 25010:2011) in the software requirements

specifications, the rule is searched again for each element of the set $\{qms_1'^{AS}, qms_2'^{AS}\}$ – the results of such a search are shown in Table 2.3.

Tab. 2.3. List of missing (after supplementing) software quality attributes in the specification and the results of the method of forming a logical conclusion about the sufficiency of quality information (according to ISO 25010) in the specifications of software requirements

Missing attribute	Number of the rule	Results of the method (rule)
Operation Time	2	$fcr'=1$, $fa'=1$, $ma'=1$, $av'=1$, $rvb'=1$, $tb'=1$, $ru'=1$, $lb'=1$, $ob'=1$, $md'=1$, $mfb'=1$, $tst'=1$, $cf'=1$, $ig'=1$, $cex'=1$, $io'=1$, $ab'=1$; $fy'=2$, $ry'=3$, $ey'=2$, $uy'=2$, $my'=3$, $sy'=2$, $cy'=2$, $py'=1$; mas[Operation Time]=17/138
Number of Test Cases	13	$ma'=2$, $ft'=1$, $tst'=2$, $cf'=2$, $ig'=2$; $ry'=5$, $my'=4$, $sy'=4$; mas[Number Of Test Cases]=5/138

According to rule №139, it was found that the specification has insufficient attributes for determining the certain software quality sub-characteristics (indicating these sub-characteristics), but comparative analysis of the list of sub-characteristics that are still impossible to determine after supplementing the specification, with a list of sub-characteristics that could not be determined on the basis of the initial version of the specification requirements, showed that it is possible to determine Functional completeness, Capacity, Appropriateness recognisability, Analyzability, Replaceability of the concrete software project.

According to Part 1 of Rule No. 140, it was found that the software requirements specification has still insufficient attributes for all eight software quality characteristics.

Therefore, the absence in the specification of software requirements of the other two attributes "Operation Time", "Number Of Test Cases" still leaves it impossible to calculate 18 sub-characteristics of quality, all 8 quality characteristics of software with a high level of veracity (there is still an insufficiency of information). Therefore, the subsystem for assessing the sufficiency of quality information in the software requirements specifications based on a comparative analysis of ontology again displays the conclusion to the user: "Information of specification is insufficient for determining the software quality according to ISO 25010".

After establishing the fact of insufficiency of the information of the specification of requirements for determining the quality of the software according to ISO 25010:2011,

according to 2-nd and 3-rd parts of rule No. 140, sorting of array `mas` on descending values of elements (weights of missing attributes) is performed and the indices of those elements of the sorted array `mas` are displayed, which have a value other than 0 – Table 2.4.

Tab. 2.4. List of missing (after supplementing) software quality attributes in the specification in descending order of weights

Missing quality attributes (indexes of elements of array `mas`)	**Weights (values of elements of array `mas`)**
Operation Time	17/138
Number of Test Cases	5/138

Therefore, the subsystem for assessing the sufficiency of quality information in the software requirements specifications based on a comparative analysis of ontology concluded: “For increasing the sufficiency of quality information in the specification, it is recommended to supplement the attributes in the following sequence: 1) Operation Time; 2) Number Of Test Cases".

Next, the sufficiency of the amount of information available in the requirements specification for assessing the quality of the software was assessed:

- by sub-characteristics:

$$q'^{AS}_{schr} = \frac{0}{4}+\frac{1}{5}+\frac{1}{6}+\frac{2}{14}+\frac{1}{4}+\frac{1}{5}+\frac{1}{7}+\frac{1}{7}+\frac{1}{14}+\frac{0}{5}+\frac{0}{6}+\frac{1}{8}+\frac{1}{13}+\frac{0}{11}+\frac{0}{6}+$$
$$+\frac{0}{5}+\frac{1}{7}+\frac{0}{6}+\frac{0}{6}+\frac{1}{8}+\frac{2}{6}+\frac{2}{10}+\frac{2}{8}+\frac{0}{2}+\frac{0}{2}+\frac{0}{1}+\frac{1}{4}+\frac{1}{5}+\frac{1}{11}+\frac{0}{4}+\frac{0}{3}=3{,}11,$$

$$D'^{AS}_{schr} = \frac{31-3{,}11}{31} = 0{,}90;$$

- by characteristics:

$$q'^{AS}_{chr} = \frac{2}{15}+\frac{5}{30}+\frac{2}{26}+\frac{2}{49}+\frac{4}{33}+\frac{4}{23}+\frac{2}{9}+\frac{1}{18}=0{,}99,$$

$$D'^{AS}_{chr} = \frac{8-0{,}99}{8} = 0{,}88.$$

Thus, the subsystem for assessing the sufficiency of information on quality in the specifications of software requirements based on a comparative analysis of ontologies displays the conclusion to the user: "Sufficiency of the volume of information is 90% for software quality evaluation by sub-characteristics and 88% for software quality evaluation by characteristics".

The gain of the sufficiency of the amount of information in the specification of requirements for assessing the quality of software after supplementing the specification of requirements with 2 necessary attributes:

- by sub-characteristics:

$$\Delta q_{schr}^{AS} = q_{schr}^{AS} - q'^{AS}_{schr} = 6{,}5 - 3{,}11 = 3{,}39\,,$$

$$\Delta D_{schr}^{AS} = D'^{AS}_{schr} - D_{schr}^{AS} = 0{,}90 - 0{,}79 = 0{,}11\,,$$

$$\Delta D_{schr}^{AS} = \frac{\Delta q_{schr}^{AS}}{31} = \frac{3{,}39}{31} = 0{,}11\,;$$

- by characteristics:

$$\Delta q_{chr}^{AS} = q_{chr}^{AS} - q'^{AS}_{chr} = 1{,}95 - 0{,}99 = 0.96\,,$$

$$\Delta D_{chr}^{AS} = D'^{AS}_{chr} - D_{chr}^{AS} = 0{,}88 - 0{,}76 = 0{,}12\,,$$

$$\Delta D_{chr}^{AS} = \frac{\Delta q_{chr}^{AS}}{8} = \frac{0{,}96}{8} = 0{,}12\,.$$

The process of supplementing the requirements specification continues until it is possible to determine all the sub-characteristics and quality characteristics or until it is concluded that there is insufficient information to determine the quality of the software. The customer of the software agent for improving the security of computer systems' software decided that further supplementing the specification is not economically feasible, so it was concluded that there is insufficient information on quality (attributes), and sufficiency of information volume is 90% for software quality evaluation by sub-characteristics and 88% for software quality evaluation by characteristics.

If the developer listened to the recommendations on the priority of adding the necessary attributes and added at least the most important of the missing attribute "Operation time" (1-st in the sorted list), it would be possible to define 13 more software quality sub-characteristics, and also 5 software quality characteristics. As a result, the sufficiency of quality information in the specification of requirements would significantly increase, and the size of the knowledge gap in assessing software quality would be further reduced. However, as the stated sufficiency of the quality information in the requirements specification satisfied the customer, the process of finalizing the requirements specification was stopped.

Assessment of the sufficiency of quality information (for metric analysis) in the software requirements specifications [74-76, 142-144]. During the experiment, *the specification of requirements for a software agent for improving the security of computer systems' software* was analyzed. The software quality indicators required for determining the software quality sub-characteristics and characteristics available in a particular specification have been identified. Based on the method of generating and filling the template of ontology

for determining the quality of concrete software (based on the metric information), an ontology for determining the quality of the concrete software (based on the metric information) is developed, which is formed by components for determining: software project comlexity (Fig. 2.52), software complexity (Fig. 2.53), software project quality (Fig. 2.54), software quality (Fig. 2.55).

Comparison of the developed ontology for the software agent for improving the security of computer systems' software (Fig. 2.52-2.55) with fragments of the base ontology for the subject area "Software Engineering", part "Software Quality. Metric Analysis" (Fig. 2.27-2.30) made it possible to find that in the developed ontology of the concrete software project there are no 9 indicators (Fig. 2.56): «Control Variables», «Cost Of One Line», «Project Duration», «Project Type», «Quantity Of Code Lines», «Quantity Of Links Of Each Module», «Quantity Of Modules», «Share Of Design Stage In Life cycle», «Total Quantity Of Operators», then the set of missing indicators:

$$\{ sqcxi_1^{AS},...,sqcxi_9^{AS} \} = \begin{Bmatrix} "ControlVar\ \ iables","CostOfOneL\ \ ine","ProjectDur\ \ ation", \\ "ProjectTyp\ \ e","QuantityOf\ \ CodeLines", \\ "QuantityOf\ \ LinksOfEac\ \ hModule","QuantityOf\ \ Modules", \\ "ShareOfDes\ \ ignStageIn\ \ Lifecycle","TotalQuant\ \ ityOfOpera\ \ tors" \end{Bmatrix}.$$

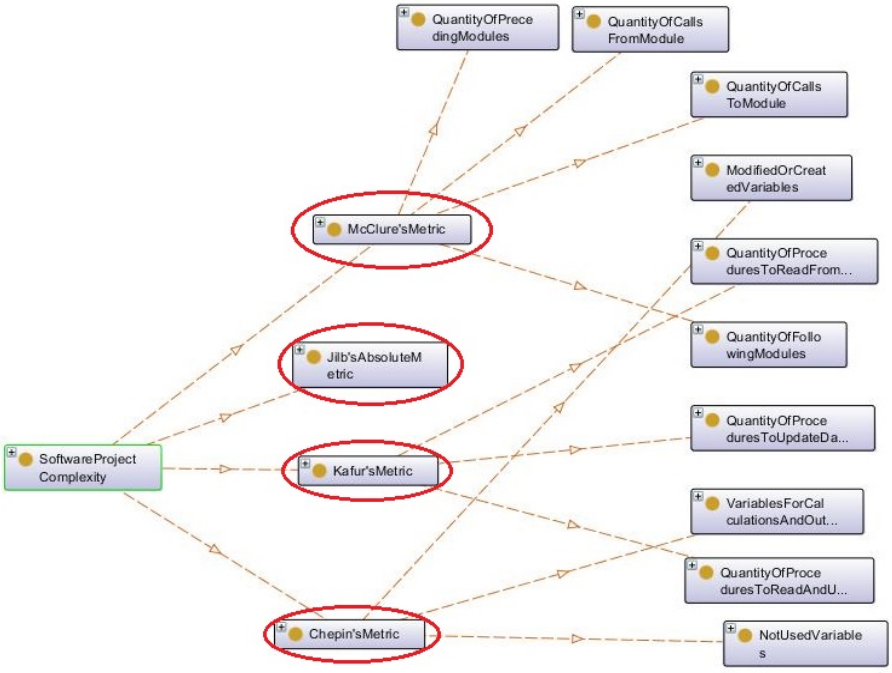

Fig. 2.52. Component of ontology for software project complexity for a software agent for improving the security of computer systems' software

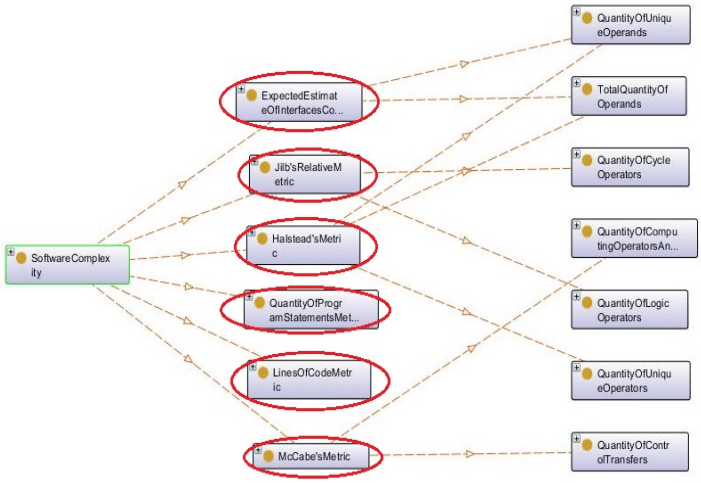

Fig. 2.53. Component of ontology for software complexity for a software agent for improving the security of computer systems' software

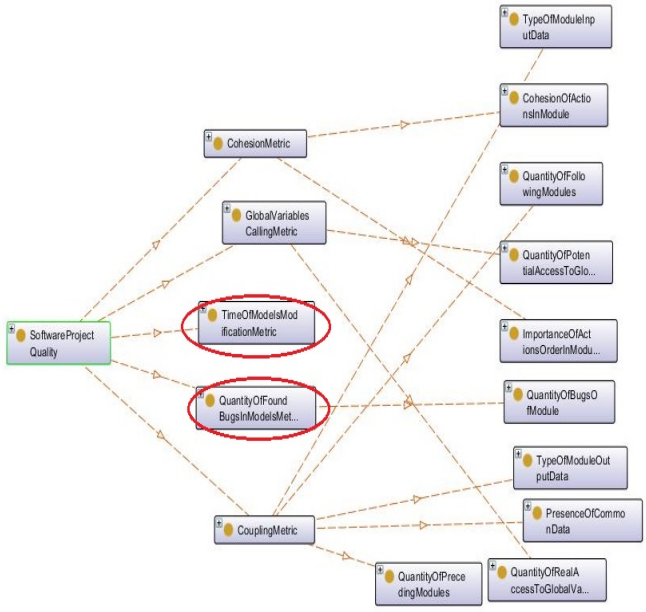

Fig. 2.54. Component of ontology for software project quality for a software agent for improving the security of computer systems' software

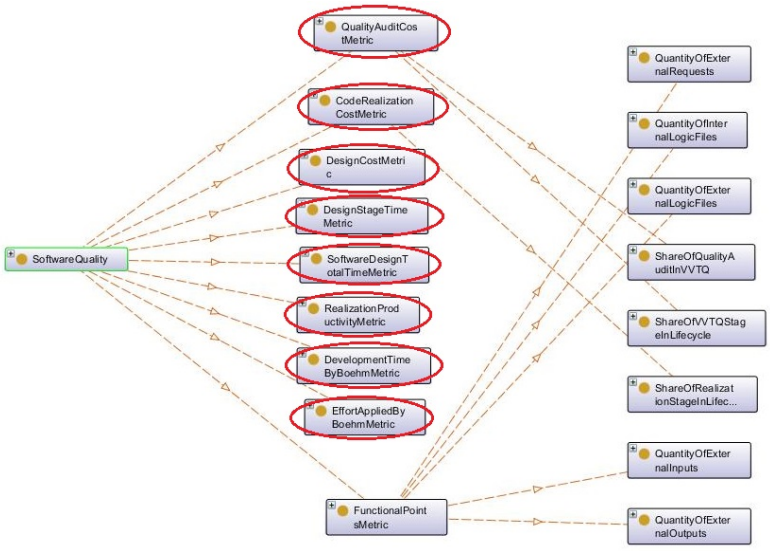

Fig. 2.55. Component of ontology for software quality for a software agent for improving the security of computer systems' software

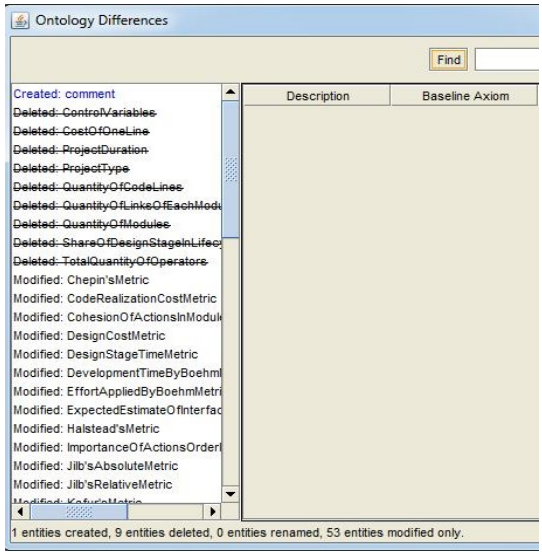

Fig. 2.56. Comparison of the ontology for determining the quality of concrete software (software agent for improving the security of computer systems' software) with the base ontology of the subject area "Software Engineering" (part "Software Quality. Metric Analysis")

According to the method of forming a logical conclusion about the sufficiency of information on quality (for metric analysis) in the software requirements specifications, a search of the production rule is made for each element of the set $\{sqcxi_1^{AS},...,sqcxi_9^{AS}\}$, according to which the indicators, which are missing in the specification for determining the software metrics, are calculated, – the results of such a search are shown in Table 2.5.

Tab. 2.5. List of software quality indicators missing in the specification and the results of the method of forming a logical conclusion about the sufficiency of information on quality (for metric analysis) in the specifications

Missing indicator	**Number of the rule**	**Results of the method (rule)**
Quantity of Modules	2	$chm=1$, $jam=1$, $mcm=1$, $km=1$, $qfbm=1$, $eicm=1$; masmt[Quantity Of Modules]=6/42
Control Variables	4	$chm=2$; masmt[Control Variables]=1/42
Quantity of Links of Each Module	6	$jam=2$; masmt[Quantity Of Links Of Each Module]=1/42
Quantity of Code Lines	21	$tmm=1$, $loc=1$, $hm=1$, $qps=1$, $sdtt=1$, $dst=1$, $dc=1$, $qac=1$, $rp=1$, $crc=1$, $eab=1$, $dtb=1$; masmt[Quantity Of Code Lines]=12/42
Project Duration	22	$tmm=2$, $sdtt=2$, $dst=2$, $rp=2$; masmt[Project Duration]=4/42
Share of Design Stage in Life Cycle	23	$tmm=3$, $dst=3$; masmt[Share Of Design Stage In Life cycle]=2/42
Total Quantity of Operators	27	$hm=2$, $mc=1$, $jrm=1$, $qps=2$; masmt[Total Quantity Of Operators]=4/42
Cost of One Line	33	$dc=2$, $qac=2$, $crc=2$; masmt[Cost Of One Line]=3/42
Project Type	42	$eab=2$, $dtb=2$; masmt[Project Type]=2/42

According to 1-st part of the rule №43, it was found that the specification of the requirements for the software agent for improving the security of computer systems' software

has insufficient indicators for 20 metrics, for the determination of which there are insufficient indicators in the specification, are circled in Fig. 2.52-2.55.

Thus, the absence of nine indicators in the software requirements led to the impossibility of calculating 20 (from 24) software metrics, to the impossibility of calculating software project and software complexity and quality with a high level of veracity and, accordingly, to the impossibility of determining the future software quality with a high level of veracity. Then the subsystem for assessing the sufficiency of quality information in the software requirements specifications based on a comparative analysis of ontologies concluded: "The information in this specification is insufficient for determining the quality of software based on the metric analysis".

After establishing the fact of insufficiency of information on quality (indicators) in the specification of requirements to the software agent for improving the security of computer systems' software, according to 2-nd and 3-rd parts of rule No.43, sorting of array `masmt` n descending values of elements (weights of missing indicators) is performed and the indices of those elements of the sorted array `masmt` are displayed, which have a value other than 0 – Table 2.6.

Tab. 2.6. List of software quality indicators missing in the specification in descending order of weights

Missing quality indicator (indexes of elements of array `masmt`)	**Weights (values of elements of array `masmt`)**
Quantity of Code Lines	12/42
Quantity of Modules	6/42
Project Duration	4/42
Total Quantity of Operators	4/42
Cost of One Line	3/42
Project Type	2/42
Share of Design Stage in Life cycle	2/42
Control Variables	1/42
Quantity of Links of Each Module	1/42

Table 2.6 shows the priority and order of consideration and supplementing the quality indicators to the specification of requirements of a software agent for improving the security of computer systems' software. Therefore, the subsystem for assessing the sufficiency of quality information in the software requirements specifications based on a comparative analysis of ontology concluded: “For increasing the sufficiency of quality information in the specification, it is recommended to supplement the indicators in the following sequence: 1) Quantity Of Code Lines; 2) Quantity Of Modules; 3) Project Duration; 4) Total Quantity Of Operators; 5) Cost Of One Line; 6) Project Type; 7) Share Of Design Stage In Life cycle; 8) Control Variables; 9) Quantity Of Links Of Each Module".

Next, the sufficiency of the amount of information (indicators) available in the requirements specification for assessing the quality of the software was assessed:

$$q_{metr}^{AS} = \frac{2}{5}+\frac{2}{2}+\frac{1}{5}+\frac{1}{4}+\frac{0}{2}+\frac{0}{5}+\frac{0}{2}+\frac{3}{3}+\frac{1}{2}+\frac{1}{1}+\frac{2}{5}+\frac{1}{3}+\frac{1}{3}+\frac{2}{2}+\frac{1}{3}+\frac{2}{2}+$$
$$+\frac{3}{3}+\frac{2}{2}+\frac{2}{4}+\frac{2}{2}+\frac{2}{3}+\frac{0}{5}+\frac{2}{2}+\frac{2}{2}=13{,}92,$$

$$D_{metr}^{AS} = \frac{24-13{,}92}{24} = 0{,}42.$$

Thus, the subsystem for assessing the sufficiency of information on quality in the specifications of software requirements based on a comparative analysis of ontologies displays the conclusion to the user: "Sufficiency of the volume of information (indicators) is 42% for software quality evaluation".

Because the proposed methods for assessing the sufficiency of quality information (for metric analysis) in the software requirements specifications based on ontology and weighted ontology are iterative, and for this software project there are metrics for determination of which there are insufficient indicators in the specification, a request was made to supplement the software requirements specification. After supplementing the requirements specification, an ontology (version 2) for determining the quality of concrete software was re-developed.

A comparative analysis of the developed ontology (version 2) with fragments of the base ontology of the subject area "Software Engineering" (part "Software Quality. Metric Analysis") found that the specification of software requirements was supplemented by indicators "Quantity Of Modules" (2-nd in the sorted list), "Total Quantity Of Operators" (4-th in the sorted list), then the set of missing indicators

$$\{ sqcx\, i_1'^{AS}, ..., sqcx\, i_7'^{AS} \} = \begin{Bmatrix} "ControlVar\ \ iables", "CostOfOneL\ \ ine", "ProjectDur\ \ ation", \\ "ProjectTyp\ \ e", "QuantityOf\ \ CodeLines", \\ "QuantityOf\ \ LinksOfEac\ \ hModule", \\ "ShareOfDes\ \ ignStageIn\ \ Lifecycle" \end{Bmatrix}.$$

According to the method of forming a logical conclusion about the sufficiency of information on quality (for metric analysis) in the software requirements specifications, the rule is searched again for each elements of the set $\{ sqcxi_1'^{AS}, ..., sqcxi_7'^{AS} \}$ – the results of such a search are shown in Table 2.7.

According to 1-st part of the rule No. 139, it was found that the specification has insufficient indicators for determining the certain software metrics (indicating these metrics), but comparative analysis of the list of metrics that are still impossible to determine after supplementing the specification, with a list of metrics that could not be determined on the basis of the initial version of the specification requirements, showed that it is possible to determine some metrics. Therefore, the absence in the specification of software requirements of the other seven attributes still leaves it impossible to calculate 14 software metrics with a high level of

veracity (there is still an insufficiency of information). Therefore, the subsystem for assessing the sufficiency of quality information in the software requirements specifications based on a comparative analysis of ontology again displays the conclusion to the user: "Information of specification is insufficient for determining the software quality based on the metric analysis".

Tab. 2.7. List of missing (after supplementing) software quality indicators in the specification and the results of the method of forming a logical conclusion about the sufficiency of quality information (for metric analysis) in the specifications of software requirements

Missing indicator	**Number of the rule**	**Results of the method (rule)**
Control Variables	4	$chm' = 1$; masmt[Control Variables]=1/42
Quantity of Links of Each Module	6	$jam' = 1$; masmt[Quantity Of Links Of Each Module]=1/42
Quantity of Code Lines	21	$tmm' = 1$, $loc' = 1$, $hm' = 1$, $qps' = 1$, $sdtt' = 1$, $dst' = 1$, $dc' = 1$, $qac' = 1$, $rp' = 1$, $crc' = 1$, $eab' = 1$, $dtb' = 1$; masmt[Quantity Of Code Lines]=12/42
Project Duration	22	$tmm' = 2$, $sdtt' = 2$, $dst' = 2$, $rp' = 2$; masmt[Project Duration]=4/42
Share of Design Stage in Life cycle	23	$tmm' = 3$, $dst' = 3$; asmt[Share Of Design Stage In Life cycle]=2/42
Cost of One Line	33	$dc' = 2$, $qac' = 2$, $crc' = 2$; masmt[Cost Of One Line]=3/42
Project Type	42	$eab' = 2$, $dtb' = 2$; masmt[Project Type]=2/42

After establishing the fact of insufficiency of the information (indicators) of the specification of requirements for determining the quality of the software, according to 2-nd and 3-rd parts of rule No. 43, sorting of array masmt on descending values of elements (weights of missing indicators) is performed and the indices of those elements of the sorted array masmt are displayed, which have a value other than 0 – Table 2.8.

Therefore, the subsystem for assessing the sufficiency of quality information in the software requirements specifications based on a comparative analysis of ontology concluded: “For increasing the sufficiency of quality information in the specification, it is recommended to supplement the attributes in the following sequence: 1) Quantity Of Code Lines; 2) Project

Duration; 3) Cost Of One Line; 4) Project Type; 5) Share Of Design Stage In Life cycle; 6) Control Variables; 7) Quantity Of Links Of Each Module".

Tab. 2.8. List of missing (after supplementing) software quality indicators in the specification in descending order of weights

Missing quality indicator (indexes of elements of array `masmt`)	**Weights (values of elements of array `masmt`)**
Quantity of Code Lines	12/42
Project Duration	4/42
Cost of One Line	3/42
Project Type	2/42
Share of Design Stage in Life cycle	2/42
Control Variables	1/42
Quantity of Links of Each Module	1/42

Next, the sufficiency of the amount of information (indicators) available in the requirements specification (after supplementing) for assessing the quality of the software was assessed:

$$q'^{AS}_{metr} = \frac{1}{5}+\frac{1}{2}+\frac{0}{5}+\frac{0}{4}+\frac{0}{2}+\frac{0}{5}+\frac{0}{2}+\frac{3}{3}+\frac{0}{2}+\frac{1}{1}+\frac{1}{5}+\frac{0}{3}+\frac{0}{3}+\frac{1}{2}+\frac{0}{3}+\frac{2}{2}+ \\ +\frac{3}{3}+\frac{2}{2}+\frac{2}{4}+\frac{2}{2}+\frac{2}{3}+\frac{0}{5}+\frac{2}{2}+\frac{2}{2}=10{,}57,$$

$$D'^{AS}_{metr} = \frac{24-10{,}57}{24} = 0{,}56$$

Thus, the subsystem for assessing the sufficiency of information on quality in the specifications of software requirements based on a comparative analysis of ontologies displays the conclusion to the user: "Sufficiency of the volume of information (indicators) is 56% for software quality evaluation ".

The gain of the sufficiency of the amount of information in the specification of requirements for assessing the quality of software after supplementing the specification of requirements with 2 necessary indicators:

$$\Delta q^{AS}_{metr} = q^{AS}_{metr} - q'^{AS}_{metr} = 13{,}92 - 10{,}57 = 3{,}35,$$

$$\Delta D^{AS}_{metr} = D'^{AS}_{metr} - D^{AS}_{metr} = 0{,}56 - 0{,}42 = 0{,}14,$$

$$\Delta D^{AS}_{metr} = \frac{\Delta q^{AS}_{metr}}{24} = \frac{3{,}35}{24} = 0{,}14.$$

The customer of the software agent for improving the security of computer systems'

software decided that further supplementing the specification is not economically feasible, so it was concluded that there is insufficient information on quality (indicators), and sufficiency of information volume is 56% for software quality evaluation.

If the developer listened to the recommendations on the priority of adding the necessary attributes and added at least the most important of the missing attribute " Quantity Of Code Lines " (1-st in the sorted list), it would be possible to define 3 more software metrics. As a result, the sufficiency of quality information in the specification of requirements would significantly increase, and the size of the knowledge gap in assessing software quality would be further reduced. However, as the stated sufficiency of the quality information in the requirements specification satisfied the customer, the process of finalizing the requirements specification was stopped.

2.6. Conclusions

Software quality analysis according to ISO 25010:2011, as well as metric software quality analysis are effective tools of assessing software quality, provided that there is sufficient information for their implementation. The knowledge of experienced professionals on the interaction and correlation of characteristics and sub-characteristics by attributes, as well as on the interaction and correlation of metrics by indicators is valuable, so they should be stored and used when evaluating software requirements specifications for the sufficiency of quality information.

In the chapter the theoretical bases of using ontologies for assessing the sufficiency of quality information in software requirements specifications and the criterion of sufficiency of quality information in software requirements specifications are developed, which are the basis for developing models of information sufficiency assessment process for determining software quality based on ISO 25010:2011 and using the results of the metric analysis.

The models of the subject area "Software Engineering" (part "Software Quality"), models of the subject area "Software Engineering" (part "Software Quality. Metric Analysis") on the basis of ontologies and weighted ontologies were developed, which, based on ontologies for reflection of knowledge on the interaction and correlation of characteristics and sub-characteristics by attributes, metrics by indicators, allow to perform modeling of the process of assessing the sufficiency of information on quality in the specifications of software requirements.

Models of the process of assessing the sufficiency of information for determining the quality of software based on ISO 25010 and with using metric analysis results were developed for the formalization of the process of assessing the sufficiency of quality information in the specification of software requirements. The developed models provide a theoretical basis for the development of methods and tools for assessing the sufficiency of information on quality in the specifications of software requirements. The main contribution of the developed models is that the models make it possible to assess the availability of quality information in the specifications of the requirements for the concrete software (whether it is present or not), to identify which information is insufficient, and to determine

the recommended sequence of supplementing insufficient information.

The ontological models of the subject area "Software Engineering" (parts "Specification of software requirements (quality indicators)" and "Specification of software requirements (quality attributes)") were developed, which consider the specification of software requirements from the point of view of quality information. The developed models provide an opportunity to formalize the specification of software requirements in terms of software quality assessment. They are templates for developing the software requirements specifications in the context of software quality determination, which can be processed by both specialists and automated tools for working with specifications or intelligent agents (bots).

Developed schemes of the process of assessing the sufficiency of information for determining the quality of software based on ISO 25010:2011 and using the results of metric analysis reflect the movement of information flows in the process of assessing the sufficiency of information on quality in the software requirements specifications. The study of information flows in the process of assessing the sufficiency of information on quality in the specifications of software requirements showed the need to develop methods and tools for assessing the sufficiency of information on quality in the specifications of software requirements.

The chapter developed ontology-based methods for assessing the sufficiency of information on quality (according to ISO 25010:2011 and for metric analysis) in the specifications of software requirements, which differ from those known in that on the basis of comparative analysis of ontologies make it possible to assess the sufficiency of information (attributes and/or indicators) in the specification of software requirements for determining the certain sub-characteristics and characteristics of software quality and/or software metrics and for forming a conclusion on the need to supplement the specification of requirements with attributes and/or indicators.

Methods for assessing the sufficiency of quality information (according to ISO 25010:2011 and for metric analysis) in the software requirements specifications based on weighted ontology were developed, which differs from those known in that by marking the weights of attributes and indicators in the base ontology allows sorting all attributes and/or quality indicators, which are missing in the specification of software requirements, in descending order of values of weights, i.e. to establish the priority of their addition to the specification of requirements.

Developed methods for assessing the sufficiency of quality information (according to ISO 25010:2011 and for metric analysis) in the specifications of software requirements based on ontologies, due to its ability to supplement the requirements specification with the necessary information, make it possible to reduce the gap in knowledge and sector with unknown information about a software system, as well as increase the sufficiency of the amount of information on quality in the specification of software requirements.

The Chapter developed the methods for generating and filling of the template of ontology for determining the quality of concrete software according to ISO 25010 and based on metric information, which, leaving in the relevant base ontology only attributes or

indicators available in the specification of requirements for concrete software, allow automated and easy to obtain ontology for determining the quality of concrete software according to ISO 25010 or based on metric information.

Methods of forming a logical conclusion about the sufficiency of information on quality (according to ISO 25010:2011 and for metric analysis) in the specifications of software requirements, which, based on the proposed production rules of forming a logical conclusion about the sufficiency of information about quality (according to ISO 25010:2011 and for metric analysis) in the software requirements specifications, provide an opportunity: to form a conclusion on the sufficiency or insufficiency of information on quality (attributes and/or indicators) in the software requirements specification; in case of insufficient information, to form conclusions for which sub-characteristics and quality characteristics and/or software metrics information is insufficient, to form a sorted (by weights) list of missing attributes and/or indicators as a recommended priority of supplementing attributes and/or indicators to software specification requirements, as well as assess the sufficiency of the available information on quality (attributes and indicators) in the specification of software requirements (before and after the supplementing) and determine the gain of the sufficiency of quality information (attributes and indicators) in the specification of software requirements (after supplementing).

A structural scheme of the information technology for assessing the sufficiency of quality information in the software requirements specifications, designed to support the assessment of software quality in the early stages of the life cycle, has been developed. The developed information technology can be used for any classes of software for which a specification of requirements is developed.

The developed information technology provides:

1) make a conclusion about the sufficiency of information on quality (attributes and indicators) in the specifications of requirements;

2) establish the priority of supplementing the specification of software requirements with attributes and/or indicators (in case of insufficient information in the specification) by forming a request to supplement requirements for software;

3) evaluate and, if necessary, increase the sufficiency of the amount of information on quality (attributes and indicators) available in the specification;

4) process quality information in the software requirements specifications by software agents (bots), without the participation of specialists, which provides the ability to automate such processes and eliminate the subjective influence of specialists, as well as the safety of this information in the software company in case of dismissal of specialists.

Thus, the proposed information technology for assessing the sufficiency of quality information in the software requirements specifications provides the customer with information to select the specification of software requirements, provides an opportunity to compare different versions of specifications, i.e. is the basis for a reasoned decision on the selection of software requirements specification taking into account the sufficiency of quality information in the specification. The developed information technology, thanks to the possibility of supplementing the specification of requirements with the necessary

information, makes it possible to increase the sufficiency of the available information on quality in the specification of requirements. This approach avoids overspending and meets customer requirements for software quality.

The subsystem for assessing the sufficiency of information on quality in the software requirements specifications was developed, which is based on a comparative analysis of ontologies and is designed to assess the early stages of the software life cycle. This subsystem provides the user with a conclusion on the sufficiency of information on quality in the specification of software requirements, on the need to supplement the specification with attributes and/or indicators, on the recommended priority of supplementing attributes and/or indicators in the specification of software requirements, and on the numerical assessment of the sufficiency of the volume of information on quality (attributes and/or indicators) available in the specification.

Studies have confirmed that the use of information technology for assessing the sufficiency of quality information in software specification specifications, even after one revision of the requirements specification, made it possible to increase the sufficiency of quality information (attributes) by 12% for a software agent for improving the security of computer systems' software, i.e. to reduce the size of the knowledge gap when assessing the quality of software.

The use of information technology for assessing the sufficiency of quality information in software specification specifications even after one revision of the requirements specification made it possible to increase the sufficiency of quality information (indicators) by 14% for a software agent for improving the security of computer systems' software, i.e. to reduce the size of a knowledge gap in assessing the quality of software.

Chapter 3. Agent-Oriented Information Technology for Assessing the Initial Stages of the Software Life Cycle

3.1. Modeling the Activity of the Ontology-Based Intelligent Agent for Assessing the Software Requirements Specifications

Sets of sub-characteristics of non-functional characteristics-components of software quality have the form [148]:

$A = \{ FCom, FCor, FAppr \}$ – the set of sub-characteristics of Functional Suitability;

$B = \{ Tb, Ru, Cc \}$ – the set of sub-characteristics of Performance Efficiency;

$C = \{ Ar, Lb, Ob, Uep, Uia, Ab \}$ – the set of sub-characteristics of Usability;

$D = \{ Mat, Avb, Ft, Rcv \}$ – the set of sub-characteristics of Reliability;

$E = \{ Ce, Ib \}$ – the set of sub-characteristics of Compatibility;

$F = \{ Conf, Int, Nr, Acb, Auth \}$ – the set of sub-characteristics of Security;

$G = \{ Mod, Rub, Anb, Mdfb, Tsb \}$ – the set of sub-characteristics of Maintainability;

$H = \{ Adb, Inb, Rpb \}$ – the set of sub-characteristics of Portability.

Sets of attributes for sub-characteristics of non-functional characteristics-components of software quality have the form:

$I = \{ Nof, Ficn, Faq, Fic \}$ – the set of attributes of Functional Completeness; $J = \{ Ot, Nic, Ndi, Ca, Pc \}$ – the set of attributes of Functional Correctness; $K = \{ Ot, Nof, Ficn, Faq, Fic, Pc \}$ – the set of attributes of Functional Appropriateness;

$L = \{ Ot, Nmot, Rt, Noe, Tnt, Tskt, Mathr \}$ – the set of attributes of Time Behaviour; $M = \{ Ot, Nofl, Noe, Niore, Uwt, Nmre, Ntre, Tcc, Iou, Nolcd, Ioll, Mmu, Mtu, Mote \}$ – the set of attributes of Resource Utilization; $N = \{ Ndi, Ncu, Cbw, Mathr, Sdb \}$ – the set of attributes of Capacity;

$O = \{ Nof, Nott, Niodi, Cnd, Fua, Uaio \}$ – the set of attributes of Appropriateness Recognisability; $P = \{ Nof, Ot, Efl, Nmot, Hfq, Eudhs, Haa, Cudhf \}$ – the set of attributes of Learnability; $Q = \{Nof, Ot, Ecr, Nsf, Nuec, Nac, Niore, Nop, Niwccvd, Nmi, Nie, Pha, Neum\}$ – the set of attributes of Operability; $R = \left\{ \begin{matrix} Nurs, Nuec, Otpdo, Nouheo, Niewusc, \\ Nacie, Necwusc, Tnect, Nfiuet, Tnfrtc, Tniop \end{matrix} \right\}$ – the set of attributes of User Error Protection; $S = \{Nie, Nige, Dipu, Disu, Dea, Drwmu\}$ – the set

of attributes of User Interface Aesthethics; $T = \{Ewscbuusd, Ewusd, Ffrusd, Susd, Ppsa\}$ – the set of attributes of Accessibility;

$U = \begin{Bmatrix} Ot, Noft, Nofl, Ps, Ntc, Nrf, Ncf, \\ Fdatc, Frn, Frl, Mtbf, Tmy, Elfd, Fdy \end{Bmatrix}$ – the set of attributes of Maturity;

$V = \{Ot, Ttdwsis, Nob, Tdt\}$ – the set of attributes of Availability; $W = \{Nofl, Ntc, Nbd, Nof, Nio\}$ – the set of attributes of Fault Tolerance; $X = \{Ot, Nbd, Ttr, Dt, Nrs, Nrn, Ray\}$ – the set of attributes of Recoverability;

$Y = \{Ot, Nofl, Nof, Ndi\}$ – the set of attributes of Co-Existence; $Z = \{Ot, Ndfrt, Ndfbe, Nip, Deay\}$ – the set of attributes of Interoperability;

$AA = \{Ot, Nio, Ntc, Nidc, Ndi, Nat, Ncr, Aca, Ndiced, Ndibred\}$ – the set of attributes of Confidentiality; $AB = \{Ot, Nio, Ntc, Nidc, Ndi, Nat, Ncr, Aca\}$ – the set of attributes of Integrity; $AC = \{Nepuds, Nernrp\}$ – the set of attributes of Non-Repudiation; $AD = \{Nasdrsl, Naao\}$ – the set of attributes of Accountability; $AE = \{Npam\}$ – the set of attributes of Authenticity;

$AF = \{Ot, Nofl, Nrf, Nmm, Nv, Nof, Nm\}$ – the set of attributes of Modularity; $AG = \{Fcy, Nfcy, Vrn, Aay, Tay, Cra\}$ – the set of attributes of Reusability; $AH = \{Nofl, Ndi, Ert, Nirbl, Ndfr, Atc\}$ – the set of attributes of Analysability; $AI = \{Ot, Nrv, Nrf, Ert, Nof, Ccca, Ntcpbm, Ntspam\}$ – the set of attributes of Modifability; $AJ = \{Ot, Ntc, Nrf, Nbtfr, Ntdos, Ncp\}$ – the set of attributes of Testability;

$AK = \begin{Bmatrix} Nof, Ot, Noft, Puf, Ndi, Nds, \\ Aads, Hea, Sea, Noftwnca, Tnfwtde \end{Bmatrix}$ – the set of attributes of Adaptability;

$AL = \{Noft, Nso, Nis, Eoi\}$ – the set of attributes of Installability; $AM = \{Nof, Ndi, Net\}$ – the set of attributes of Replaceability.

Taking into account the obtained sets of attributes and sub-characteristics of non-functional characteristics-components of software quality, the obtained in Chapter 2 functions of dependences of characteristics on sub-characteristics and sub-characteristics on attributes, as well as the relationship between ontology concepts "depends on", we develop *base (universal) ontological models of non-functional characteristics-components of software quality*. Then:

- base ontological model of Functional Suitability:

$$\begin{aligned} O_{Fs} &= \{\{ Fs, A, I, J, K \}, "depends\ \ on", \{ f_1, \varphi_1, \varphi_2, \varphi_3 \}\} = \\ &= \{\{ fsa_1, ..., fsa_{19} \}, "depends\ \ on", \{ f_1, \varphi_1, \varphi_2, \varphi_3 \}\} \end{aligned}, \quad (3.1)$$

where $fsa_1 = Fs$, $\{fsa_2, ..., fsa_4\} \in A$, $\{fsa_5, ..., fsa_8\} \in I$, $\{fsa_9, ..., fsa_{13}\} \in J$, $\{fsa_{14}, ..., fsa_{19}\} \in K$;

- base ontological model of Performance Efficiency:

$$O_{Pe} = \{\{ Pe, B, L, M, N \}, "depends\ on", \{ f_2, \varphi_4, \varphi_5, \varphi_6 \}\} = \\ = \{\{ pea_1, ..., pea_{30} \}, "depends\ on", \{ f_2, \varphi_4, \varphi_5, \varphi_6 \}\}; \quad (3.2)$$

- base ontological model of Usability:

$$O_{Ub} = \{\{ Ub, C, O, P, Q, R, S, T \}, "depends\ on", \\ \{ f_3, \varphi_7, \varphi_8, \varphi_9, \varphi_{10}, \varphi_{11}, \varphi_{12} \}\} = \\ = \{\{ uba_1, ...uba_{56} \}, "depends\ on", \{ f_3, \varphi_7, \varphi_8, \varphi_9, \varphi_{10}, \varphi_{11}, \varphi_{12} \}\}; \quad (3.3)$$

- base ontological model of Reliability:

$$O_{Rb} = \{\{Rb, D, U, V, W, X\}, "depends\ on", \{f_4, \varphi_{13}, \varphi_{14}, \varphi_{15}, \varphi_{16}\}\} = \\ = \{\{rba_1, ..rba_{35}\}, "depends\ on", \{f_4, \varphi_{13}, \varphi_{14}, \varphi_{15}, \varphi_{16}\}\}; \quad (3.4)$$

- base ontological model of Compatibility:

$$O_{Cb} = \{\{ Cb, E, Y, Z \}, "depends\ on", \{ f_5, \varphi_{17}, \varphi_{18} \}\} = \\ = \{\{ cba_1, ..., cba_{12} \}, "depends\ on", \{ f_5, \varphi_{17}, \varphi_{18} \}\}; \quad (3.5)$$

- base ontological model of Security:

$$O_{Scr} = \{\{ Scr, F, AA, AB, AC, AD, AE \}, "depends\ on", \\ \{ f_6, \varphi_{19}, \varphi_{20}, \varphi_{21}, \varphi_{22}, \varphi_{23} \}\} = \\ = \{\{ scra_1, ..., scra_{29} \}, "depends\ on", \{ f_6, \varphi_{19}, \varphi_{20}, \varphi_{21}, \varphi_{22}, \varphi_{23} \}\}; \quad (3.6)$$

- base ontological model of Maintainability:

$$O_{Mb} = \{\{ Mb, G, AF, AG, AH, AI, AJ \}, "depends\ on", \\ \{ f_7, \varphi_{24}, \varphi_{25}, \varphi_{26}, \varphi_{27}, \varphi_{28} \}\} = \\ = \{\{ mba_1, ..., mba_{39} \}, "depends\ on", \{ f_7, \varphi_{24}, \varphi_{25}, \varphi_{26}, \varphi_{27}, \varphi_{28} \}\}; \quad (3.7)$$

- base ontological model of Portability:

$$O_{Pb} = \{\{Pb, H, AK, AL, AM\}, "depends\ on", \{f_8, \varphi_{29}, \varphi_{30}, \varphi_{31}\}\} = \\ = \{\{pba_1, ..., pba_{22}\}, "depends\ on", \{f_8, \varphi_{29}, \varphi_{30}, \varphi_{31}\}\}. \quad (3.8)$$

Ontologies (ontological knowledge bases), which are developed according to models (3.1) – (3.8), are filled on the basis of information taken from the standards ISO 25010:2011, ISO 25023:2016.

Then *ontological models of non-functional characteristics-components of quality of concrete software* have a form:

- ontological model of Functional Suitability:

$$O_{Fs}^{real} = \{\{fsa_1,...,fsa_m\},"depends\ on",\{f_1,\varphi_1,\varphi_2,\varphi_3\}\}, \quad (3.9)$$

where $m \le 19$ – the number of Functional Suitability sub-characteristics available in the specification of the concrete software requirements, as well as the number of Functional Suitability sub-characteristics that can be calculated based on the available attributes;

- ontological model of Performance Efficiency:

$$O_{Pe}^{real} = \{\{pea_1,..., pea_n\},"depends\ on",\{f_2,\varphi_4,\varphi_5,\varphi_6\}\}; \quad (3.10)$$

where $n \le 30$ – the number of Performance Efficiency sub-characteristics available in the specification of the concrete software requirements, as well as the number of Performance Efficiency sub-characteristics that can be calculated based on the available attributes;

- ontological model of Usability:

$$O_{Ub}^{real} = \{\{uba_1,...uba_k\},"depends\ on",\{f_3,\varphi_7,\varphi_8,\varphi_9,\varphi_{10},\varphi_{11},\varphi_{12}\}\}; \quad (3.11)$$

where $k \le 56$ – the number of Usability sub-characteristics available in the specification of the concrete software requirements, as well as the number of Usability sub-characteristics that can be calculated based on the available attributes;

- ontological model of Reliability:

$$O_{Rb}^{real} = \{\{rba_1,...rba_o\},"depends\ on",\{f_4,\varphi_{13},\varphi_{14},\varphi_{15},\varphi_{16}\}\}; \quad (3.12)$$

where $o \le 35$ – the number of Reliability sub-characteristics available in the specification of the concrete software requirements, as well as the number of Reliability sub-characteristics that can be calculated based on the available attributes;

- ontological model of Compatibility:

$$O_{Cb}^{real} = \{\{cba_1,...,cba_p\},"depends\ on",\{f_5,\varphi_{17},\varphi_{18}\}\}; \quad (3.13)$$

where $p \le 12$ – the number of Compatibility sub-characteristics available in the specification of the concrete software requirements, as well as the number of Compatibility sub-characteristics that can be calculated based on the available attributes;

- ontological model of Security:

$$O_{Scr}^{real} = \{\{scra_1,...,scra_q\},"depends\ on",\{f_6,\varphi_{19},\varphi_{20},\varphi_{21},\varphi_{22},\varphi_{23}\}\}; \quad (3.14)$$

where $q \le 29$ – the number of Security sub-characteristics available in the specification of the concrete software requirements, as well as the number of Security sub-characteristics that can be calculated based on the available attributes;

- ontological model of Maintainability:

$$O_{Mb}^{real} = \{\{mba_1,...,mba_r\},"depends\ on",\{f_7,\varphi_{24},\varphi_{25},\varphi_{26},\varphi_{27},\varphi_{28}\}\}; \quad (3.15)$$

where $r \le 39$ – the number of Maintainability sub-characteristics available in the specification of the concrete software requirements, as well as the number of Maintainability sub-characteristics that can be calculated based on the available attributes;

- ontological model of Portability:

$$O_{Pb}^{real} = \{\{pba_1,...,pba_l\},"depends\ on",\{f_8,\varphi_{29},\varphi_{30},\varphi_{31}\}\}, \quad (3.16)$$

where $l \le 22$ – the number of Portability sub-characteristics available in the specification of the concrete software requirements, as well as the number of Portability sub-characteristics that can be calculated based on the available attributes.

Ontologies (ontological knowledge bases), which are developed according to models (3.9) – (3.16), are filled on the basis of information taken from the specification of requirements for the concrete software.

The proposed ontology-based intelligent agent (OBIA)uses during its operation the base ontologies of non-functional characteristics-components of software quality as known facts with which it compares the information obtained from the specification of requirements for the concrete software, presented in the form of real ontologies. On the basis of such comparison, OBIA assesses the information in the specification of software requirements and decides on further actions.

Thus, *the process of assessing the specification of requirements by an intelligent agent* is to: 1) compare ontologies of non-functional characteristics-components of the quality of the concrete software with base ontologies of non-functional characteristics of software for identifying the attributes, missing in the specification of requirements for concrete software, on that real ontologies were developed, and for identifying the sub-characteristics and non-functional characteristics of the software, which cannot be calculated on the basis of the attributes, available in the requirements for the concrete software; 2) forming a conclusion on the sufficiency or insufficiency of information in the specification of requirements for determining each non-functional characteristic of the software separately and for determining all non-functional characteristics of the software together; 3) calculation of numerical assessments of the level of sufficiency of information available in the specification of requirements for determining each non-functional characteristics of the software; 4) calculation of the numerical assessment of the level of sufficiency of the information available in the specification of requirements for determination of all non-functional characteristics of the software.

Let $SMM_{Fs} = O_{Fs} \setminus (O_{Fs} \cap O_{Fs}^{real})$, where: $SMM_{Fs} = \{fsa_1,..., fsa_{(19-m)}\}$ – the set of attributes of sub-characteristics of Functional Suitability that are missing in the specification of requirements for the concrete software, as well as the number of sub-

characteristics of Functional Suitability that cannot be calculated on the basis of available attributes; $SMM_{Pe} = O_{Pe} \setminus (O_{Pe} \cap O_{Pe}^{real})$, where: $SMM_{Pe} = \{pea_1,..., pea_{(30-n)}\}$ – the set of attributes of sub-characteristics of Performance Efficiency that are missing in the specification of requirements for the concrete software, as well as the number of sub-characteristics of Performance Efficiency that cannot be calculated on the basis of available attributes; $SMM_{Ub} = O_{Ub} \setminus (O_{Ub} \cap O_{Ub}^{real})$, where: $SMM_{Ub} = \{uba_1,..., uba_{(56-k)}\}$ – the set of attributes of sub-characteristics of Usability that are missing in the specification of requirements for the concrete software, as well as the number of sub-characteristics of Usability that cannot be calculated on the basis of available attributes; $SMM_{Rb} = O_{Rb} \setminus (O_{Rb} \cap O_{Rb}^{real})$, where $SMM_{Rb} = \{rba_1,..., rba_{(35-o)}\}$ – the set of attributes of sub-characteristics of Reliability that are missing in the specification of requirements for the concrete software, as well as the number of sub-characteristics of Reliability that cannot be calculated on the basis of available attributes; $SMM_{Cb} = O_{Cb} \setminus (O_{Cb} \cap O_{Cb}^{real})$, where: $SMM_{Cb} = \{cba_1,..., cba_{(12-p)}\}$ – the set of attributes of sub-characteristics of Compatibility that are missing in the specification of requirements for the concrete software, as well as the number of sub-characteristics of Compatibility that cannot be calculated on the basis of available attributes; $SMM_{Scr} = O_{Scr} \setminus (O_{Scr} \cap O_{Scr}^{real})$, where: $SMM_{Scr} = \{scra_1,..., scra_{(29-q)}\}$ – the set of attributes of sub-characteristics of Security that are missing in the specification of requirements for the concrete software, as well as the number of sub-characteristics of Security that cannot be calculated on the basis of available attributes; $SMM_{Mb} = O_{Mb} \setminus (O_{Mb} \cap O_{Mb}^{real})$, where: $SMM_{Mb} = \{mba_1,..., mba_{(39-r)}\}$ – the set of attributes of sub-characteristics of Maintainability that are missing in the specification of requirements for the concrete software, as well as the number of sub-characteristics of Maintainability that cannot be calculated on the basis of available attributes; $SMM_{Pb} = O_{Pb} \setminus (O_{Pb} \cap O_{Pb}^{real})$, where: $SMM_{Pb} = \{pba_1,..., pba_{(22-l)}\}$ – the set of attributes of sub-characteristics of Portability that are missing in the specification of requirements for the concrete software, as well as the number of sub-characteristics of Portability that cannot be calculated on the basis of available attributes.

Production rules for forming the conclusion on the sufficiency or insufficiency of information in the specification of requirements for determining each non-functional characteristic of the software:

- for Functional Suitability:

$$\begin{array}{c} if\ SMM_{Fs} = \varnothing \\ then\ "SRS\ information\ is\ sufficient\ for\ Functional\ Suitability", \\ else\ "SRS\ information\ is\ insufficient\ for\ Functional\ Suitability" \end{array} \quad (3.17)$$

- for Performance Efficiency:

$$\begin{array}{c} if\ SMM_{Pe} = \varnothing \\ then\ "SRS\ information\ is\ sufficient\ for\ Performance\ Efficiency", \\ else\ "SRS\ information\ is\ insufficient\ for\ Performance\ Efficiency" \end{array} \quad (3.18)$$

- for Usability:

$$\begin{array}{c} if\ SMM_{Ub} = \varnothing \\ then\ "SRS\ information\ is\ sufficient\ for\ Usability", \\ else\ "SRS\ information\ is\ insufficient\ for\ Usability" \end{array} \quad (3.19)$$

- for Reliability:

$$\begin{array}{c} if\ SMM_{Rb} = \varnothing \\ then\ "SRS\ information\ is\ sufficient\ for\ Reliability", \\ else\ "SRS\ information\ is\ insufficient\ for\ Reliability" \end{array} \quad (3.20)$$

- for Compatibility:

$$\begin{array}{c} if\ SMM_{Cb} = \varnothing \\ then\ "SRS\ information\ is\ sufficient\ for\ Compatibility", \\ else\ "SRS\ information\ is\ insufficient\ for\ Compatibility" \end{array} \quad (3.21)$$

- for Security:

$$\begin{array}{c} if\ SMM_{Scr} = \varnothing \\ then\ "SRS\ information\ is\ sufficient\ for\ Security", \\ else\ "SRS\ information\ is\ insufficient\ for\ Security" \end{array} \quad (3.22)$$

- for Maintainability:

$$\begin{array}{c} if\ SMM_{Mb} = \varnothing\ then\ "SRS\ information\ is\ sufficient\ for\ Maintainability" \\ else\ "SRS\ information\ is\ insufficient\ for\ Maintainability" \end{array}, \quad (3.23)$$

- for Portability:

$$\begin{array}{l} if\ SMM_{Pb} = \varnothing\ then\ "SRS\ information\ is\ sufficient\ for\ Portability" \\ \quad else\ "SRS\ information\ is\ insufficient\ for\ Portability" \end{array} . \quad (3.24)$$

Production rule for forming the conclusion on the sufficiency or insufficiency of information in the specification of requirements for determining all non-functional characteristics of the software:

$$\begin{array}{l} if\ (SMM_{Fs} \cup SMM_{Pe} \cup SMM_{Ub} \cup SMM_{Rb} \cup SMM_{Cb} \cup \\ \quad SMM_{Scr} \cup SMM_{Mb} \cup SMM_{Pb}) = \varnothing \\ \quad then\ "SRS\ information\ is\ sufficient" \\ \quad else\ "SRS\ information\ is\ insufficient" \end{array} . \quad (3.25)$$

The developed model of activity of ontology-based intelligent agent for assessing the software requirements specifications reflects features of assessment of information sufficiency for determining the non-functional characteristics-components of software quality and is a theoretical basis for the development of methods of activity of OBIA for software requirements specification assessing.

3.2. Methods of Activity of the Ontology-Based Intelligent Agents for Semantic Parsing the Software Requirements Specifications

For automating the semantic analysis of a natural language specification, it is necessary to formalize it. Formalization of the specification of software requirements should be performed, taking into account for which requirements the semantic parsing of the specification will be directed. For parsing specifications for finding the attributes for determining the non-functional characteristics-components of software quality, such formalization will be performed using ontologies, because ontologies provide the visualization of duplication and gaps in knowledge based on visualization of missing logical connections, as well as the ability to analyze information by intelligent agents. For such formalization of the requirements specification, the base ontology of the software requirements specification (in terms of attributes), developed in Chapter 2 based on the ISO 29148:2018 standard, can be used. In this ontology, the attributes needed to determine the non-functional characteristics-components of the software quality are presented taking into account the distribution by sections of the specification, due to which the developed ontology is a template of software requirements specification in terms of attributes and provides visual clues to the user about the location of the attributes in the software requirements specification.

The ontology-based intelligent agent for semantic parsing the software requirements specifications accepts the software requirements specification and performs automatic

parsing of the software requirements specification in order to find the attributes needed for determining the non-functional characteristics of the software.

Method of activity of the ontology-based intelligent agent for semantic parsing the software requirements specifications in terms of search of the attributes consists of the following stages [127, 128]:

1) search for each attribute from the base ontology of the specification of requirements (such ontology is contained in the knowledge base of the agent) in the specification of requirements for the concrete software (let the specification is formalized and meets the requirements of ISO 29148:2018);

2) if $<attribute_i>$ is found in the requirements specification, then $<attribute_i>$ is entered in the set of available attributes, i = 1..138 (because, according to ISO 25023:2016, there are 138 different attributes from which the non-functional characteristics-components of software quality depend);

3) if $<attribute_i>$ is not found in the requirements specification, then $<attribute_i>$ is entered in the set of missing attributes, i = 1..138;

4) from the base ontology for non-functional characteristics-components of software quality, developed in Chapter 2, all attributes from the set of missing attributes are removed;

5) it is checked whether all the attributes from the set of available attributes remained in the ontology after its modification in the previous step;

6) there is a preservation of changes – the creation of a real ontology for non-functional characteristics-components of software quality.

Thus, the result of such an intelligent agent is a real ontology for non-functional characteristics-components of software quality, which is the input for the ontology-based intelligent agent for evaluating the initial stages of the software life cycle. As additional results of the agent's operation, for further work, the sets of available and missing attributes in the real specification of software requirements can also be used.

The structure of the ontology-based intelligent agent for semantic parsing the natural language specifications of software requirements is presented in Fig. 3.1.

The developed intelligent agent makes it possible to perform the analysis of natural language specifications to establish the presence or absence of attributes needed for determining the non-functional characteristics of the software. These results are then used to assess the sufficiency of information (attributes) for determining the non-functional characteristics-the components of software quality. Because for determining the sufficiency it's necessary to know if an attribute is present or missing in the specification, the rules for parsing the specifications on the basis of which the developed agent works are simple. The simplicity of these rules ensures high speed and low cost of parsing the natural language specifications.

By analogy, we develop *the method of activity of the ontology-based intelligent agent for semantic analysis of software requirements to search for indicators for metric analysis* – Fig. 3.2. Such an agent accepts natural language software requirements and automatically analyzes the requirements to find the indicators needed to determine the software metrics. The result of the work of this intelligent agent is a real ontology of the subject area "Software

Engineering" (part "Software Quality. Metric Analysis"). Based on this ontology, the sufficiency of information for determining the software complexity and quality metrics can be calculated according to the approach described in Chapter 2.

An intelligent agent operating on the basis of the method from Fig. 3.2 makes it possible to perform semantic analysis of natural language specifications in order to find the indicators needed to determine the metrics of complexity and quality of software. These results are then used to assess the sufficiency of the information of requirements for determining the software metrics, for which it is only necessary to establish the presence or absence of each indicator in the requirements.

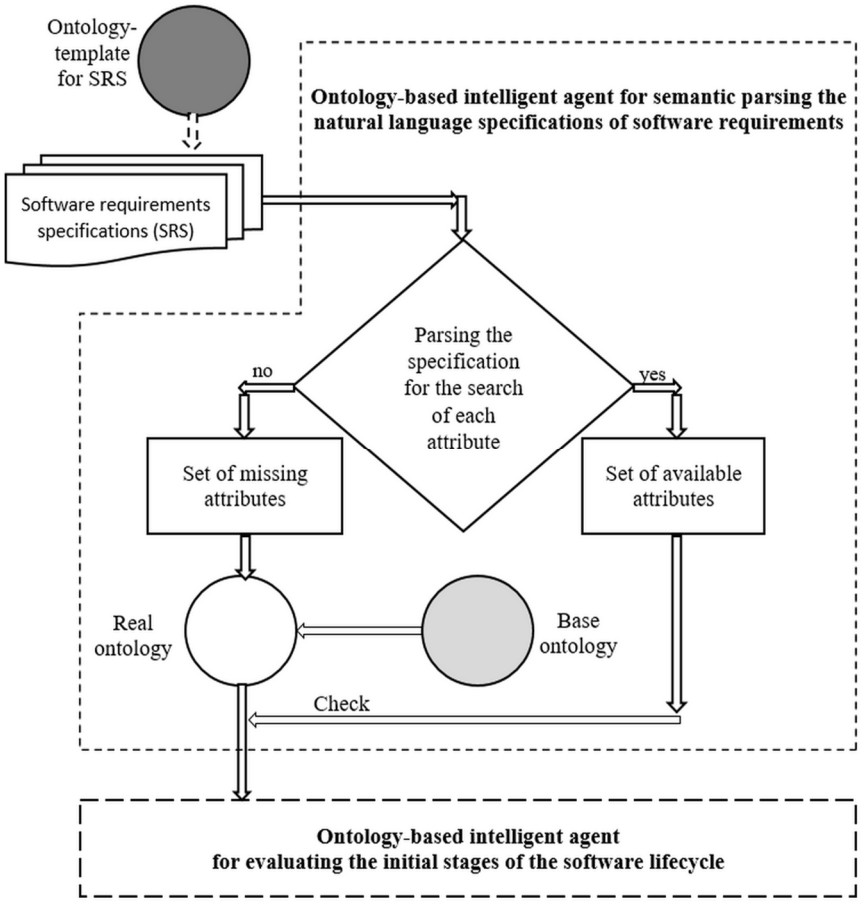

Fig. 3.1. Structure of of the ontology-based intelligent agent for semantic parsing the natural language specifications of software requirements

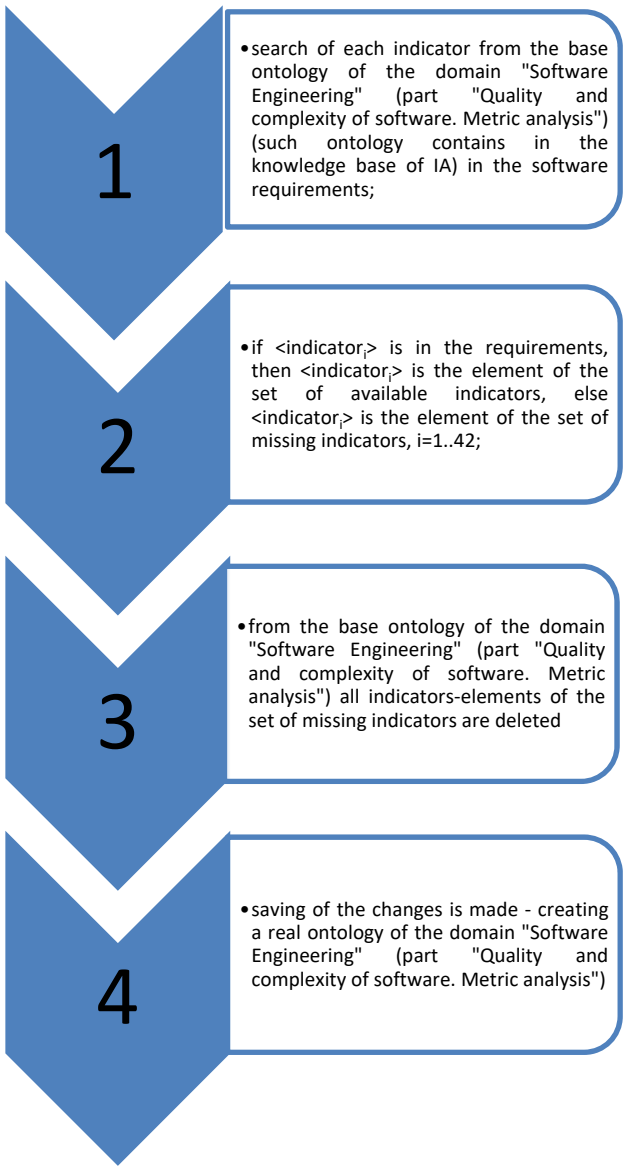

Fig. 3.2. Method of activity of intelligent agent for semantic analysis of software requirements to search for indicators for metric analysis

3.3. Methods of Activity of Ontology-Based Intelligent Agents for Evaluating the Initial Stages of the Software Life Cycle

The ontology-based intelligent agent uses during its operation the base ontologies of non-functional characteristics-components of software quality (which were developed in Chapter 2) as facts known to him. It is ontologies, which reflect the causal relationships between concepts, make it possible to identify attributes that are missing in the specification and to determine which non-functional characteristics-components of software quality cannot be determined without such attributes. With these base ontologies, the agent compares the information obtained from the specification of the requirements for the concrete software, presented in the form of real ontologies for ensuring the ability to compare base and real ontologies. Based on this comparison of ontologies, the intelligent agent obtains a list of attributes that are missing in the specification, because it is the attributes, which are missing in the specification, will distinguish real ontologies from the base ones. The intelligent agent analyzes the obtained list of missing attributes and dependencies of non-functional characteristics on attributes (according to base ontologies) and determines which non-functional characteristics-components of software quality cannot be determined without missing attributes. In addition, the intelligent agent counts the number of missing attributes and non-functional characteristics, which cannot be calculated without certain attributes, for forming a numerical assessment of the sufficiency of information in the software requirements specifications. After that, the intelligent agent assesses the information in the specification of software requirements and decides on further actions, in particular, provides conclusions on the sufficiency of information and recommendations for its improvement.

So, *method of activity of ontology-based intelligent agent for evaluating the initial stages of the software life cycle for assessing the sufficiency of information in the specification for determining the non-functional characteristics of the software quality* consists of the following stages [149-154]:

1) comparison of ontologies for non-functional characteristics-components of the quality of the concrete software with base ontologies of non-functional characteristics-components of software quality in order to identify attributes missing in the specification of requirements for the concrete software;

2) identification of sub-characteristics and non-functional characteristics-components of software quality, which cannot be calculated on the basis of the attributes available in the specification for the concrete software;

3) forming a conclusion on the sufficiency or insufficiency of information in the specification of requirements for determining each non-functional characteristic of the software separately and for determining all non-functional characteristics of the software together;

4) calculation of numerical assessments of the level of sufficiency of the information available in the specification of requirements for determining each non-functional characteristic of the software according to the formula:

$$D_j = \frac{(k_j - \sum_{i=1}^{k_j} \frac{qm_i}{qn_i})}{k_j}, \quad (3.26)$$

where k_j – the number of sub-characteristics of the j-th non-functional characteristic of the software ($j = 1..8$, because the ISO 25010 standard defines 8 non-functional characteristics-components of the software quality), qm_i – the number of attributes for the i-th sub-characteristic of the j-th non-functional characteristics of the software, qn_i – the number of required attributes for the i-th sub-characteristic of the j-th non-functional characteristics of the software (determined by the base ontologies for each non-functional characteristic-component of the software quality);

5) calculation of numerical assessment of the level of sufficiency of information available in the specification of requirements for determining all non-functional characteristics-components of software quality according to the formula:

$$D = \frac{(k - \sum_{j=1}^{k} \frac{qmc_j}{qnc_j})}{k}, \quad (3.27)$$

where k – the number of non-functional characteristics–components of software quality ($k = 8$ according to ISO 25010), qmc_j – the number of attributes missing in the specification for the concrete software for the j-th non-functional characteristic of the software, qnc_j – the number of required attributes for the j-th non-functional characteristic of the software (according to base ontologies for each non-functional characteristic-component of software quality);

6) visualization of gaps in knowledge about non-functional characteristics-components of software quality.

The proposed agent is intelligent because it automatically processes the existing knowledge (requirements on non-functional characteristics, presented in the form of ontologies) and generates new knowledge (conclusions about the level of information sufficiency, recommendations for improving the sufficiency of information in the specification of requirements).

This intelligent agent does not work with fuzzy data, because the task of assessing the sufficiency of information does not involve fuzzy data. The required attribute is either present in the specification or absent in it, and then there is a drop in the level of sufficiency of information by a value that depends on how many non-functional characteristics depend on this attribute (correlated by it).

By analogy, we develop the *ontology-based intelligent agent for evaluating the initial stages of the software life cycle for assessing the sufficiency of information in the specification for determining the software metrics* [149-154]. The base ontology for such an agent will be the ontology of the subject area "Software Engineering" (part "Software Quality. Metric Analysis"), developed in Chapter 2. In addition, the numerical assessment of the sufficiency of metric information in the specification will be calculated by the formula:

$$D = \frac{(l - \sum_{j=1}^{l} \frac{qmi_j}{qni_j})}{l}, \quad (3.28)$$

where l – number of metrics ($l = 24$, because 24 metrics of complexity and quality are available in the early stages of the life cycle), qmi_j – the number of indicators missing in the real specification for the j-th metric, qni_j – the number of required indicators for the j-th metric, which is determined by the base ontology of the subject area "Software Engineering" (part "Software Quality. Metric Analysis")), $j = 1..k$, $\Sigma(qni_j) = 72$ – the total number of indicators (including those repetitive), on which the software metrics depend.

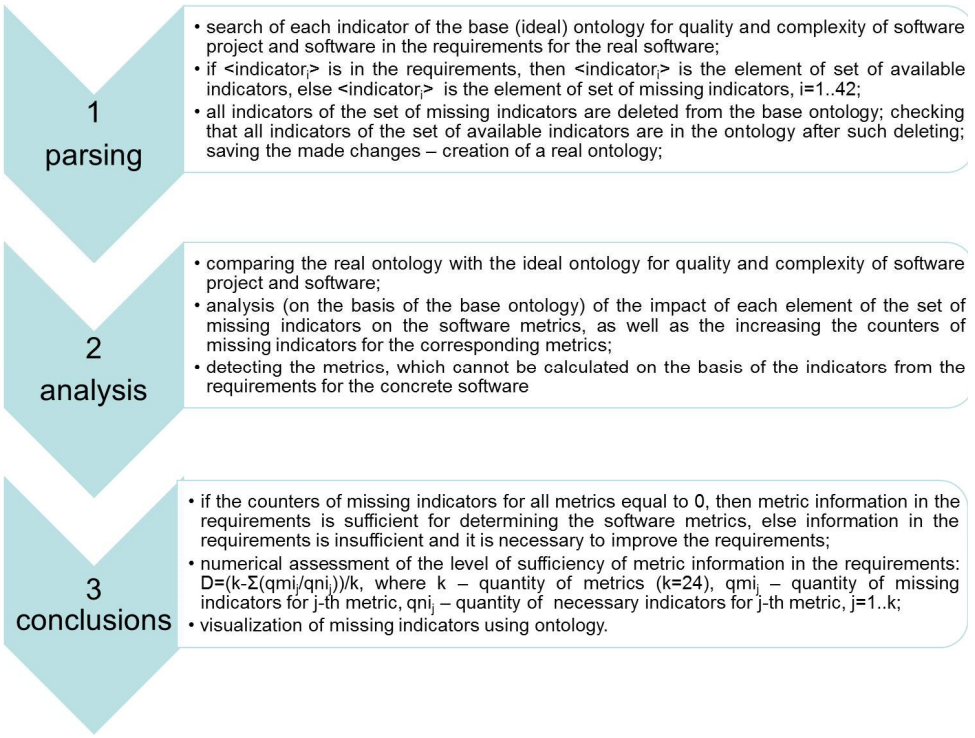

Fig. 3.3. Method of activity of OBIA for evaluating the initial stages of the software life cycle for assessing the sufficiency of information in the specification for determining the software metrics

Then the method of activity of ontology-based intelligent agent activity for evaluating the initial stages of the software life cycle for assessing the sufficiency of information in the specification for determining the software metrics is represented in Fig. 3.3.

Thus, the developed ontology-based intelligent agent for evaluating the initial stages of the software life cycle for assessing the sufficiency of information in the specification for determining the software metrics forms a conclusion about the sufficiency or insufficiency of metric information in the specification of software requirements, calculates numerical assessment of the sufficiency of metric information, as well as visualized missing indicators with a distribution of metrics for which they are used.

3.4. Agent-Oriented Information Technology for Assessing the Initial Stages of the Software Life Cycle

Agent-oriented information technology (AOIT) for assessing the initial stages of the software life cycle solves the problem of assessing the sufficiency of information of requirements for determining the non-functional characteristics of the software. According to ISO 25010, non-functional characteristics-components of software quality are Reliability, Functional Suitability, Performance Efficiency, Compatibility, Maintainability, Portability, Security, Usability. The sub-characteristics of non-functional characteristics are also defined by ISO 25010. The attributes on the basis of which non-functional characteristics and their sub-characteristics are determined are defined by ISO 25023:2016. Then the sufficiency of the information of requirements for determining non-functional characteristics is determined by the presence in the specification of all attributes required for determining the non-functional characteristics (203 attributes, including 138 different attributes).

Taking into account the theoretical and applied foundations of information technology for assessing the sufficiency of information on quality in the software requirements specifications and the movement of information flows in the process of assessing the sufficiency of information of requirements for determining the non-functional characteristics, which were developed in Chapter 2, the agent-oriented information technology for assessing the initial stages of the software life cycle is developed, that uses the tools and methods of accumulation, processing and transmission of primary information for obtaining the information of new quality about the state of the object, process or phenomenon) [77, 153, 154] – Fig. 3.4.

The purpose of agent-oriented information technology is automating the quantitative assessment of the level of sufficiency of information of requirements for determining the non-functional characteristics in order to minimize the impact of the human factor and to simplify the implementation of this assessment by both developers and customers. Information technology allows identifying the need to form a repeat request for supplementing the attributes needed for determining the non-functional characteristics, and, if it's necessary, generates it and visualizes its content.

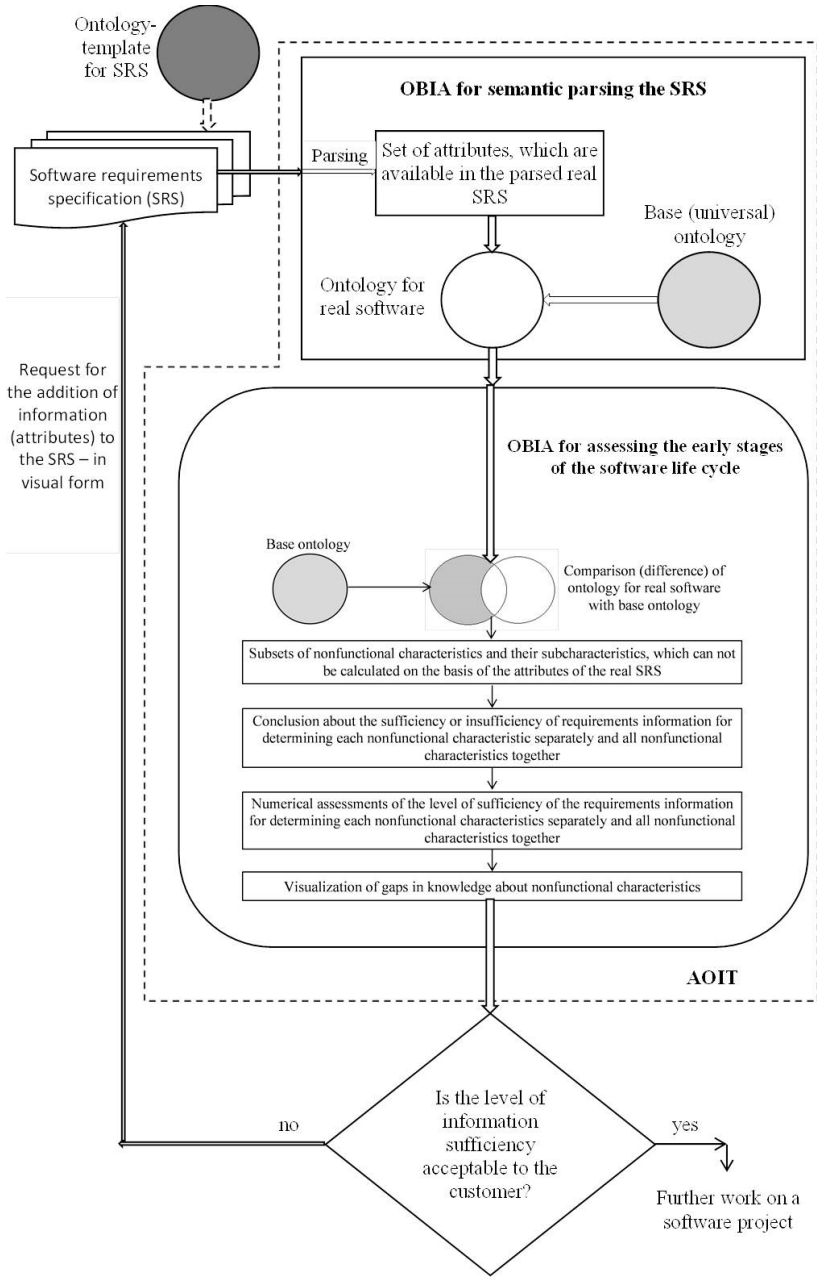

Fig. 3.4. Agent-oriented information technology for assessing the initial stages of the software life cycle

The proposed information technology automatically processes the existing knowledge (non-functional requirements of the specification) and generates new knowledge (conclusions about the sufficiency of information, the level of sufficiency of information, recommendations for improving the sufficiency of information in the specification of requirements).

As can be seen from Fig. 3.4, agent-oriented information technology is based on the ontology-based intelligent for semantic parsing the software requirements specifications and the ontology-based intelligent agent for evaluating the initial stages of the software life cycle.

The ontology-based intelligent agent for semantic parsing the software requirements specifications accepts the software requirements specification and performs automatic parsing of the software requirements specification in order to find the attributes needed for determining the non-functional characteristics of the software or indicators needed for determining the software metrics. The software requirements specification templates, which demonstrates all the necessary attributes and indicators, as well as their location in the specification, are offered to the user in the form of a base ontology of the subject area "Software Engineering" (parts "Specification of software requirements (attributes)", "Specification of software requirements (indicators)"), developed in Chapter 2 on the basis of the ISO 29148 standard. After carrying out parsing of the specification the sets of attributes and indicators available in the concrete specification and the sets of attributes and indicators missing in the concrete specification are formed. Using the base (universal) ontology developed in Chapter 2 for non-functional characteristics and for metric analysis (known facts) and the sets of available and missing attributes and indicators, the intelligent agent generates a real ontology for the concrete software, which is transmitted as input to the ontology-based intelligent agent for evaluating the initial stages of the software life cycle.

The ontology-based intelligent agent for evaluating the initial stages of the software life cycle compares the base ontologies for non-functional characteristics and for metric analysis (known facts) with the appropriate ontologies for real software. As a result of such comparison, the agent, in accordance with the developed method of activity, forms subsets of non-functional characteristics and their sub-characteristics, and sets of the metrics, which cannot be calculated on the basis of the attributes and indicators available in the real specification; provides a conclusion on the sufficiency or insufficiency of information in the specification of requirements for determining each non-functional characteristic of the software separately and for determining all non-functional characteristics of the software together; provides a conclusion on the sufficiency or insufficiency of metric information in the specification of requirements; calculates numerical assessments of the level of sufficiency of information available in the specification of requirements for determining each non-functional characteristic of the software, all non-functional characteristics, software metrics; provides visualization of gaps in knowledge in software requirements.

If the level of sufficiency of information is 100% or is acceptable to the customer, then further work on the software project is performed, otherwise, the developers of the specification are asked to supplement the necessary attributes and/or indicators to the specification (with visualized hints, which attributes/indicators must be supplement), after which the specification

can to be reworked by the developed agent-oriented information technology.

Developed agent-oriented information technology makes it possible to compare different requirements' specifications for software projects with similar or the same cost and duration, to ensure the inclusion in the requirements of information needed for further determining the non-functional characteristics and metrics, thereby reducing the gap in knowledge in requirements for software projects. The main advantage of the developed information technology is the automation of the processes of parsing the specification and assessing the sufficiency of information of requirements for determining the non-functional characteristics and software metrics, thereby eliminating the subjective human impact, as well as the safety of information in the software company in case of specialist's dismissal.

3.5. Results of Functioning the Agent-Oriented Information Technology for Assessing the Initial Stages of the Software Life Cycle

Agent-oriented information technology for assessing the initial stages of the software life cycle is implemented in PHP in the form of freely distributable software, available at the link – https://olp-project.herokuapp.com.

Before uploading the requirements' specification for its processing, the user of information technology can read the software requirements specification template, which demonstrates all the necessary attributes for determining the non-functional characteristics, as well as their location in the specification, presented as an ontology. Fig. 3.5 presents a fragment of such an ontology-template.

For performing the analysis, the user of agent-oriented information technology for assessing the initial stages of the software life cycle must upload a specification of software requirements in pdf format. After that, the information technology parsing the downloaded specification and generates an ontology for real software in owl-format, which the user can download for further work.

After comparing the ontology for real software with base ontology for non-functional characteristics, the developed AOIT gives the conclusion on the sufficiency of requirements information, which consists of: the number and percentage of missing attributes (during this counting, AOIT gives two these numbers – without and with taking into account how many times the missing attribute is used when the determination of subcharacteristics of non-functional characteristics (without and with repetitions)); quantitative assessments of the sufficiency of requirements information for determining each non-functional characteristic and all non-functional characteristics together. In addition, the developed AOIT proposes the list of missing attributes in the form of a list, which is divided by subcharacteristics of non-functional characteristics, which provides visualization of missing attributes for determining one or another subcharacteristics of non-functional characteristic.

For the experiment, three requirements specifications to software agent for improving the safety of computer systems' software, which are developed by various

software companies of Khmelnytskyi. These SRS have approximately the same cost and duration, so the choice of the SRS according to these criteria is difficult.

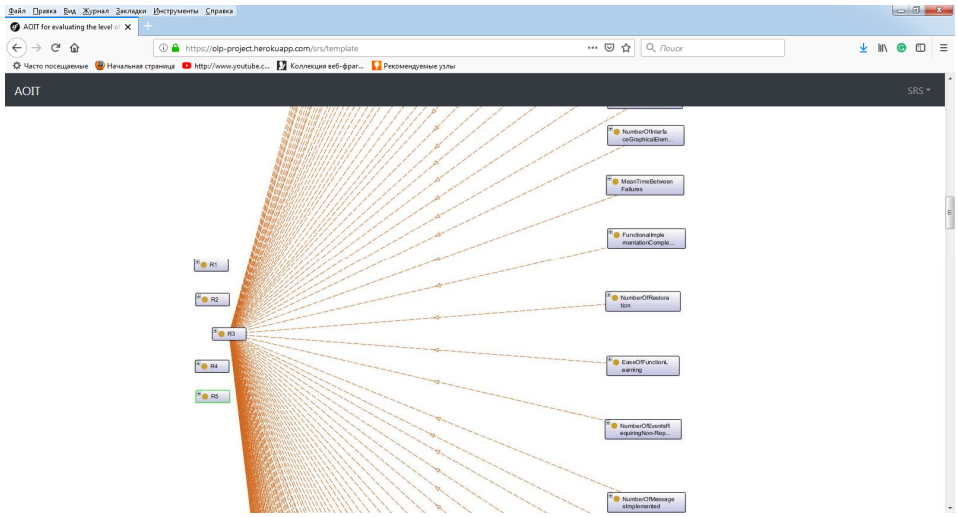

Fig. 3.5. Fragment of ontology-template for SRS of AOIT for assessing the sufficiency of information at the initial stages of the software life cycle

As a result of the analysis of SRS No. 1, the developed AOIT has provided the following conclusions – Fig. 3.6, Fig. 3.7. As a result of the analysis of SRS No. 2, the developed AOIT has provided the conclusions, which are presented in Fig. 3.8. After analysis of the SRS No. 3, the developed AOIT has provided the following conclusions – Fig. 3.9.

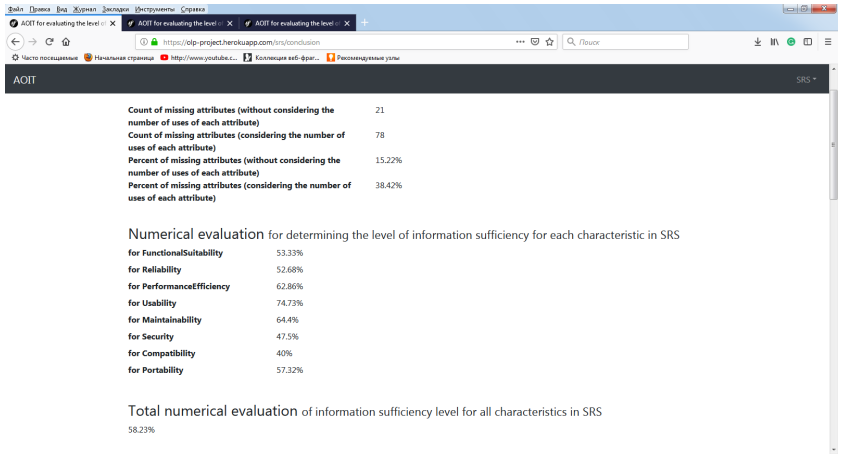

Fig. 3.6. Quantitative assessments of the sufficiency of requirements information (SRS No. 1) for determining the non-functional characteristics (which are provided by the developed AOIT)

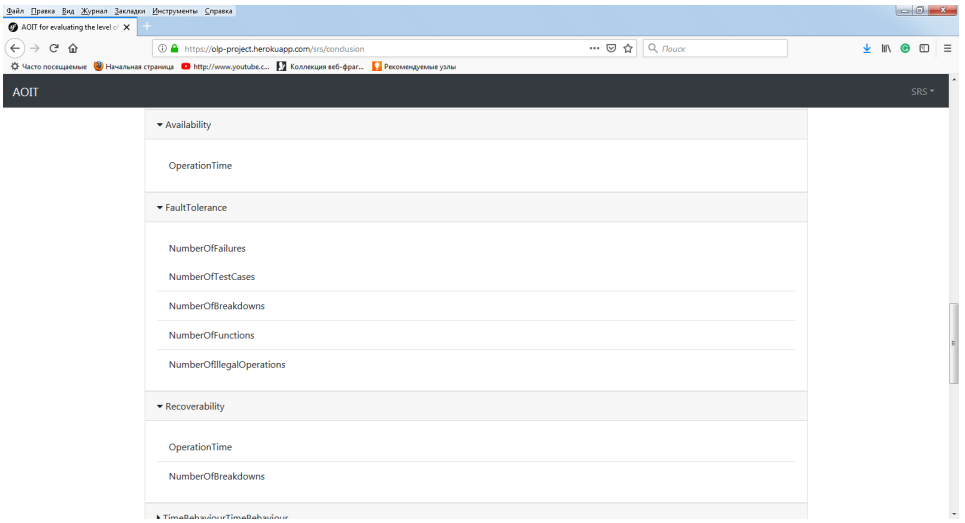

Fig. 3.7. Visualization of missing attributes (in SRS No. 1) for determining the three subcharacteristics of the non-functional characteristic "Reliability" (which is provided by the developed AOIT)

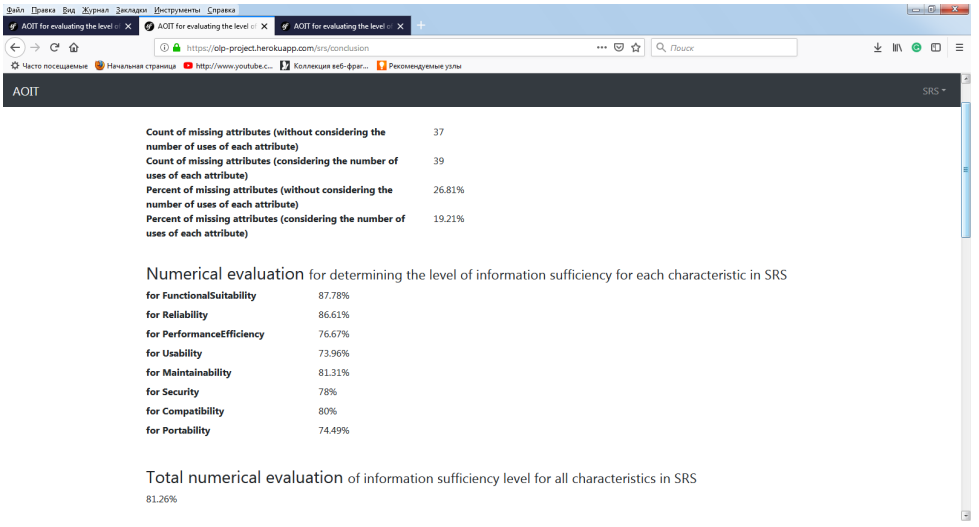

Fig. 3.8. Quantitative assessments of the sufficiency of requirements information (SRS No. 2) for determining the non-functional characteristics (which are provided by the developed AOIT)

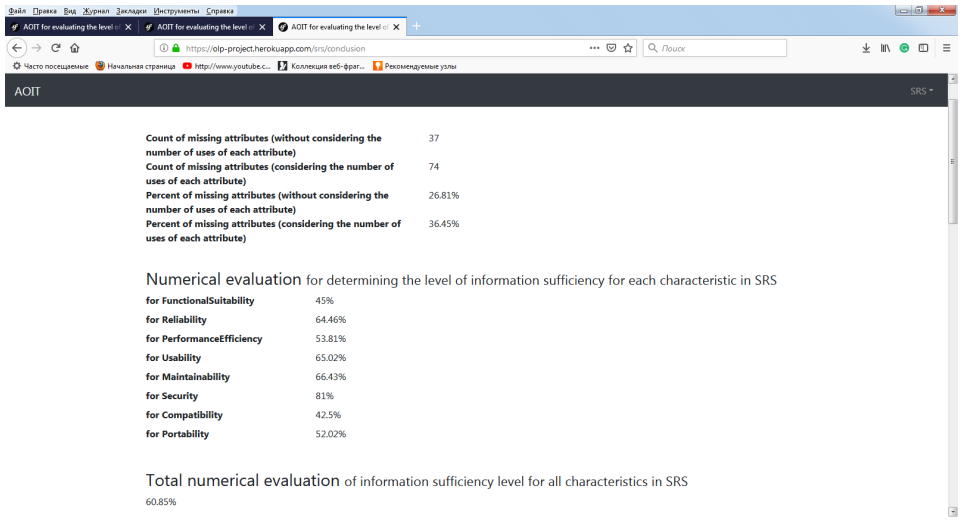

Fig. 3.9. Quantitative assessments of the sufficiency of requirements information (SRS No. 3) for determining the non-functional characteristics (which are provided by the developed AOIT)

The analysis of the results of the developed AOIT confirmed the regularity, that more important and priority are attributes, which impact on more than one subcharacteristic of non-functional characteristics. So, in SRS No. 1 there are no 21 attributes without considering the number of uses of each attribute when determining the subcharacteristics of non-functional characteristics (but there are no 78 attributes, considering the number of uses of each attribute). In SRS No. 2 there are no 37 attributes without considering the number of uses of each attribute (but there are no 39 attributes, considering the number of uses of each attribute). In SRS No. 3, there are no 37 attributes without considering the number of uses of each attribute (but there are no 74 attributes, considering the number of uses of each attribute). At the same time, the level of sufficiency of requirements information of SRS No. 1 for determining all non-functional characteristics is 58,23%, the level of sufficiency of requirements information of SRS No. 2 – 81,26%, the level of sufficiency of requirements information of SRS No. 3 – 60,85%.

Thus, with less number of missing attributes (without considering the number of uses of each attribute), SRS No. 1 has a lower level of information sufficiency than SRS No. 2, in which more attributes are absent (without considering the number of uses of each attribute). This situation is explained by the fact that in SRS No. 1 there is no greater number of attributes (in comparison with the SRS No. 2), which impact on more than one subcharacteristic of non-functional characteristics. This fact is proved by the number of missing attributes, which are provided by the AOIT (considering the number of uses of each attribute).

In the result of the analysis of the conclusions of the developed AOIT, the customer of the software agent for improving the safety of computer systems' software decided that the level of sufficiency of the requirements information in all three SRS isn't acceptable for the transition to further work on the software project. Therefore, all three SRS were sent back to developers for revision (in part of supplementing the attributes for determining the non-functional characteristics). The revised SRS were also analyzed by the developed AOIT. The conclusions of the AOIT after the analysis of the revised SRS are presented in Fig. 3.10-3.12.

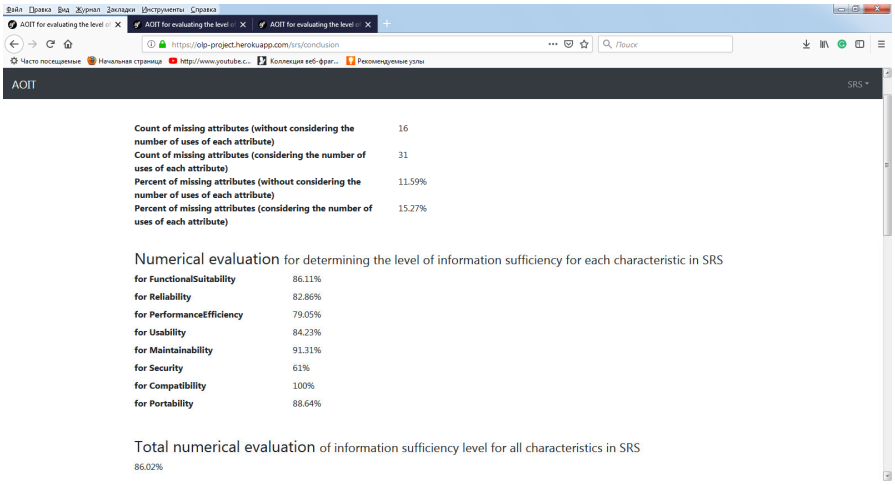

Fig. 3.10. Quantitative assessments of the sufficiency of requirements information (SRS No. 1, after revision), which are provided by the developed AOIT

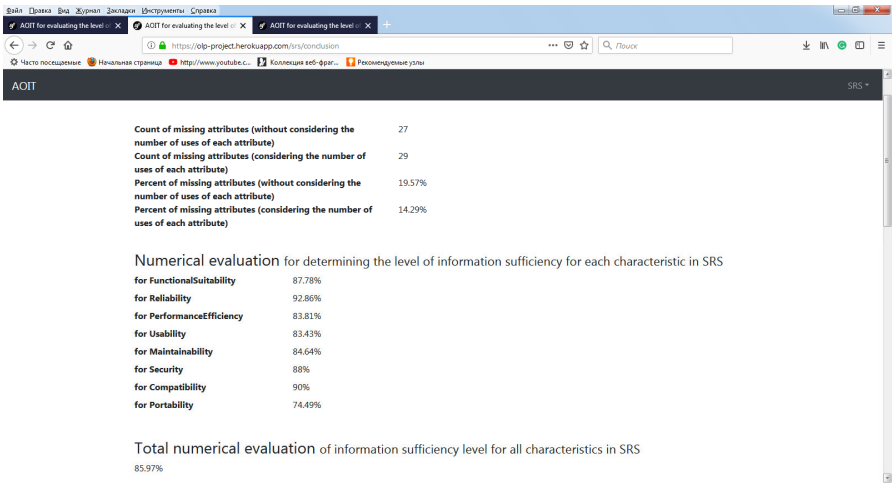

Fig. 3.11. Quantitative assessments of the sufficiency of requirements information (SRS No. 2, after revision), which are provided by the developed AOIT

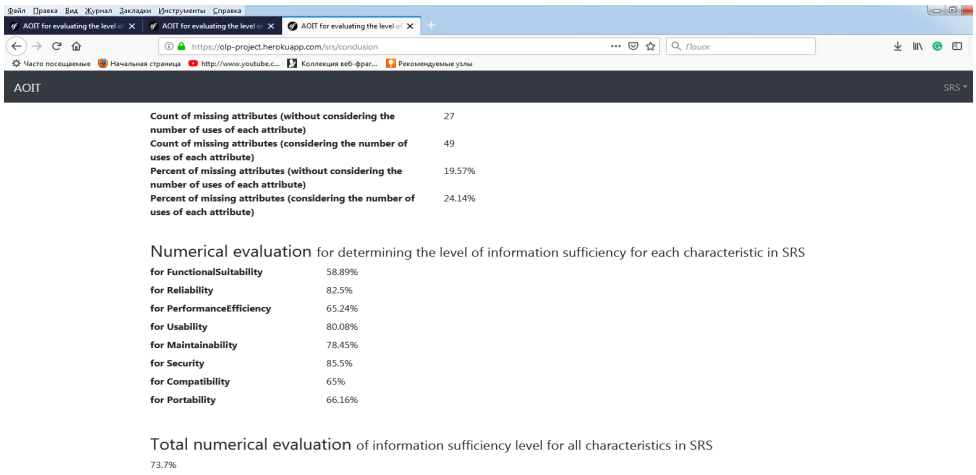

Fig. 3.12. Quantitative assessments of the sufficiency of requirements information (SRS No. 3, after revision), which are provided by the developed AOIT

In the SRS No. 1, 5 attributes were added without considering the number of uses of each attribute in determining the subcharacteristics of non-functional characteristics (47 attributes, considering the number of uses of each attribute). In SRS No. 2, 10 attributes were added without considering the number of uses of each attribute (and 10 attributes, considering the number of uses of each attribute). In SRS No. 3, 10 attributes were also added without considering the number of uses of each attribute (25 attributes, considering the number of uses of each attribute). At the same time, the level of sufficiency of requirements information of the SRS No. 1 for determining all non-functional characteristics is already 86,02%, the level of sufficiency of requirements information of the SRS No. 2 – 85,97%, the level of sufficiency of requirements information of the SRS No. 3 – 73,7%. Consequently, the conclusions of the developed AOIT for assessing the sufficiency of information at the initial stages of the software life cycle provide increasing the level of sufficiency of requirements information for determining the non-functional characteristics, respectively, by 27,79%, 4,71% and 12,85% for SRS No. 1-3.

We (the developers of AOIT) recommend the level of sufficiency equal 85% is acceptable, but the final decision regarding the required level of sufficiency is taken by the customer of the software. The customer can set a lower threshold of this level, for example, 90%, 95% or 100% (even for non-critical software). So, in the result of the analysis of the conclusions of the developed AOIT, the customer decided that the level of sufficiency of the requirements information of SRS No.1 and SRS No. 2 is already accepted for the further work on the software project. The customer selected SRS No. 1 for further work. So, the customer was able to make a choice the SRS from the view of its information's sufficiency, but not only from the view of its cost and duration.

A separate subsystem of the proposed AOIT is *the intelligent subsystem for determining the sufficiency of metric information in the software requirements specifications*. The user of the intelligent subsystem for determining the sufficiency of metric information upload specification of software requirements in pdf format. The system's agent for parsing the software requirements specification to search for metric information (indicators for calculating the metrics) pars the specification and generates a real ontology for the concrete software as a .owl file, which user can download (Fig. 3.13).

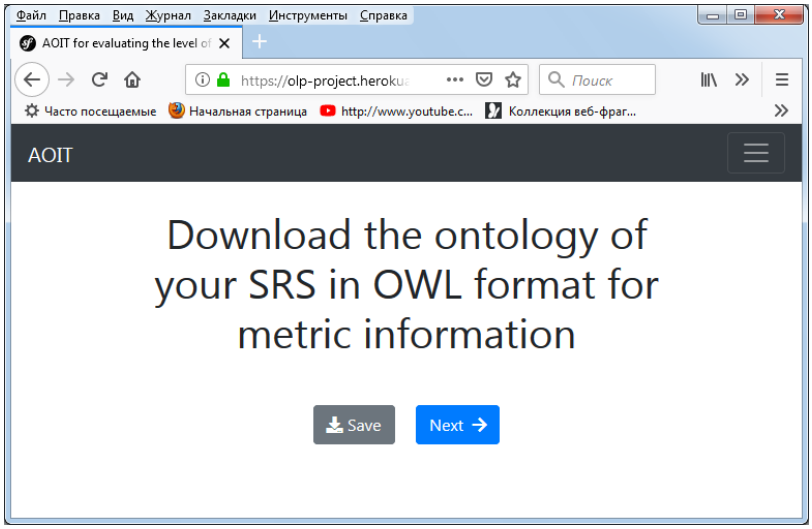

Fig. 3.13. Interface of the intelligent subsystem for determining the sufficiency of metric information in the SRS – with the possibility of download of ontology for real SRS in OWL-format

Next, the system's agent designed for assessing the sufficiency of metric information in the specifications compares the real ontology for the concrete software with the base ontology and gives the conclusion 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, as well as a visualized list of missing indicators, divided by software metrics.

For the experiment, two specifications of requirements for the software agent for improving the safety of computer systems' 'software, developed by two different software companies in Khmelnytskyi, were analyzed.

The level of sufficiency of the metric information of specification 1 was 49.72% with the missing 7 indicators (without considering the number of uses of each indicator) and 31 indicators (considering the number of uses of each indicator) – Fig. 3.14.

The level of sufficiency of the metric information of specification 2 was 75.28% with the missing 14 indicators (without considering the number of uses of each indicator) and 20 indicators (considering the number of uses of each indicator) – Fig. 3.15.

Obviously, the sufficiency of metric information in specification 2 is higher than the sufficiency of metric information in specification 1, and the number of missing indicators (considering the number of uses of each indicator) in specification 2 is lower than in specification 1 (while the number of missing indicators without considering the number of uses of each indicator in specification 2 is twice as large as in specification 1). These results are explained by the fact that more important and priority are the indicators on which more than one metric depends.

Fig. 3.16, 3.17 presents the interface of the system with the visualization of the missing indicators for determining the complexity metrics with exact values at the design stage with the distribution of metrics.

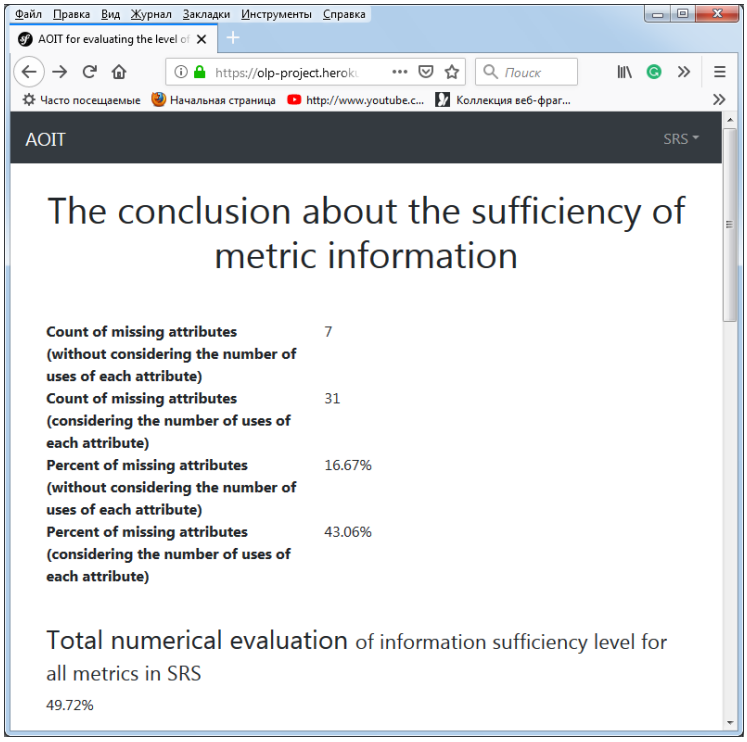

Fig. 3.14. Interface of the intelligent system for determining the sufficiency of metric information in the SRS – conclusion about the sufficiency of metric information in SRS1

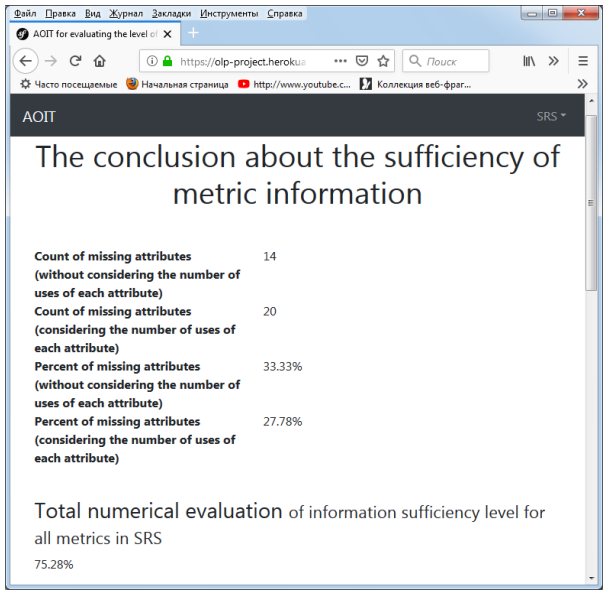

Fig. 3.15. Interface of the intelligent system for determining the sufficiency of metric information in the SRS – conclusion about the sufficiency of metric information in SRS2

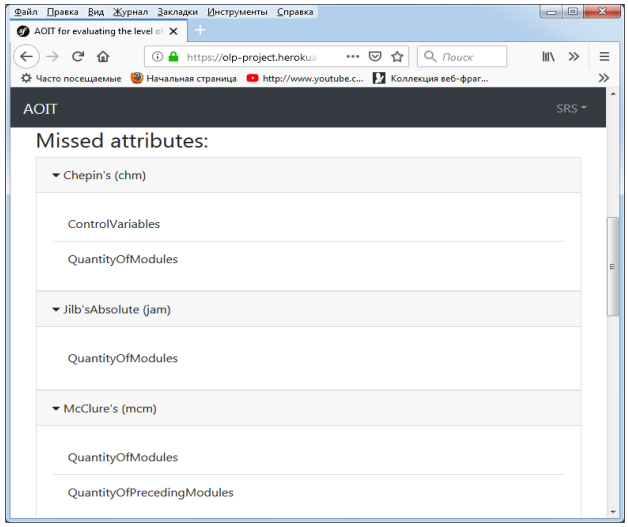

Fig. 3.16. Interface of the intelligent system for determining the sufficiency of metric information in the SRS – with the visualization of missing indicators (in SRS1) for determining the software complexity metrics with the exact values at the design stage

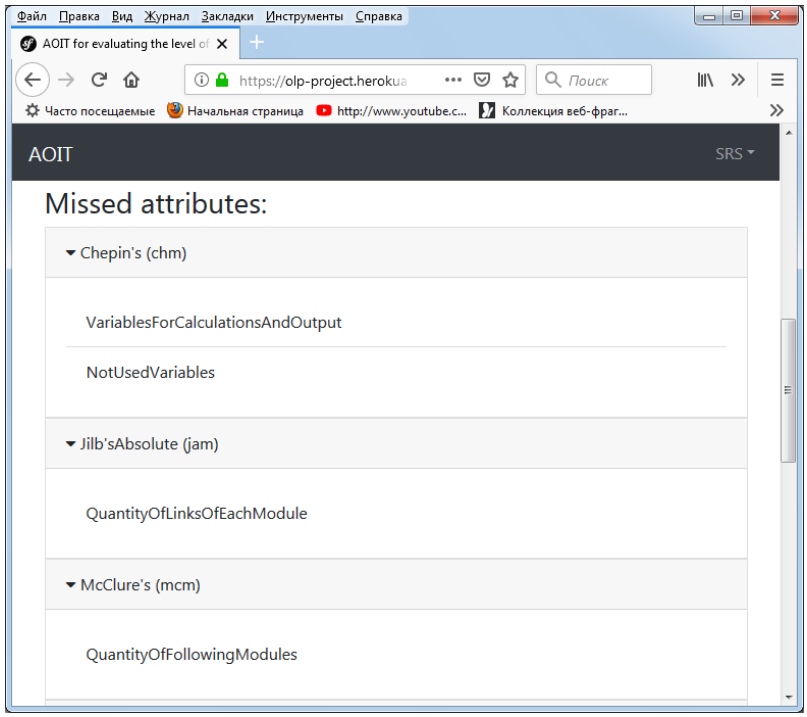

Fig. 3.17. Interface of the intelligent system for determining the sufficiency of metric information in the SRS – with the visualization of missing indicators (in SRS2) for determining the software complexity metrics with the exact values at the design stage

The customer of the software agent for improving the safety of computer systems' software considered the level of sufficiency equal 95% is acceptable (because it is software for critical use). Therefore, the customer demanded that both specifications be finalized by their developers..

After supplementing, the specifications were re-analyzed by the intelligent subsystem for determining the sufficiency of metric information in the SRS – Fig. 3.18, 3.19.

The level of sufficiency of metric information of specification 1 after re-work was 100% (the specification has all the necessary indicators for calculating metrics), and the level of sufficiency of metric information of specification 2 after re-work was 87.99% with the absence of 8 indicators (without considering the number of uses of each indicator) and 10 indicators (considering the number of uses of each indicator).

The customer of the software agent for improving the safety of computer systems' software chose specification 1 with the level of sufficiency of metric information 100% for further work on the software project.

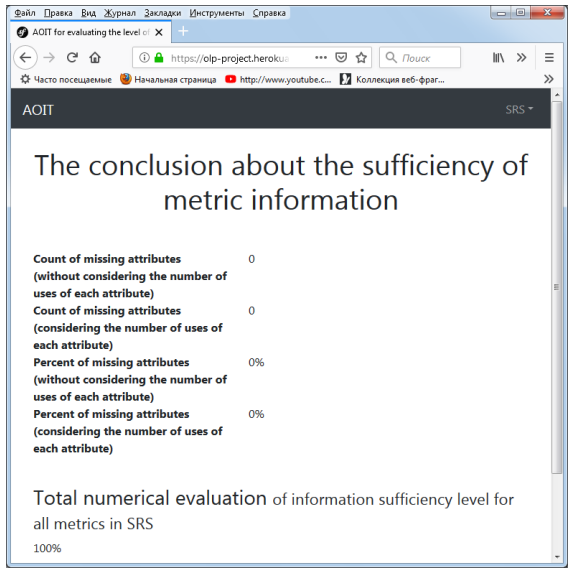

Fig. 3.18. Interface of the intelligent system for determining the sufficiency of metric information in the SRS – conclusion about the sufficiency of metric information in SRS1 (after re-work)

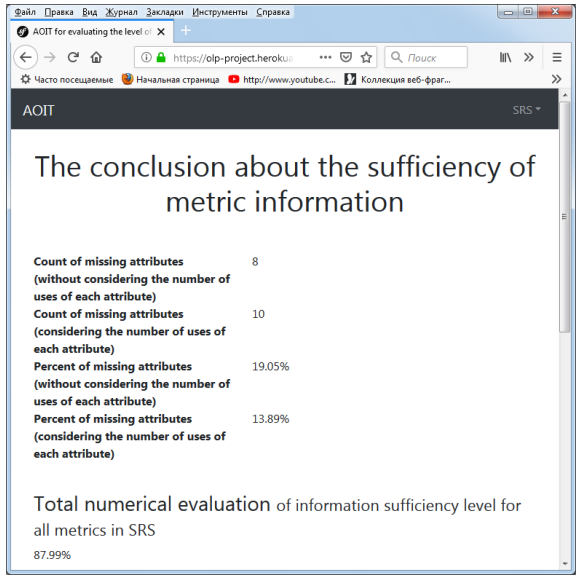

Fig. 3.19. Interface of the intelligent system for determining the sufficiency of metric information in the SRS – conclusion about the sufficiency of metric information in SRS2 (after re-work)

The developed intelligent subsystem for determining the sufficiency of metric information in the SRS provided an increase of the sufficiency of information by 50.28% for specification 1 and by 12.71% for specification 2.

3.6. Conclusions

In Chapter 3, the set-theoretical, base (universal) ontological models of non-functional characteristics-components of software quality, as well as ontological models of non-functional characteristics-components of quality of the concrete software are proposed, which are based on ISO 25010:2011 and ISO 25023:2016 and provide the basis for selecting a sufficient amount of information for assessing the non-functional characteristics of the software.

A model of activity of ontology-based intelligent agent for assessing the software requirements specifications is developed, which is based on comparative analysis of ontologies and is a theoretical basis for developing the methods of activity of ontology-based intelligent agents.

For automating the semantic analysis of the natural language specification in order to verify the compliance of its non-functional requirements with the needs of the customer, it is necessary to formalize it, for example, using ontologies. For such formalization, it was proposed to use the developed in Chapter 2 ontology for software requirements specifications, the ontology for non-functional characteristics-components of software quality, as well as the ontology for metric analysis, which became the basis (known facts) of intelligent agents for semantic parsing the natural language specifications for search of attributes and/or indicators needed for determining the non-functional characteristics-components of software quality and software metrics.

The intelligent agents for semantic parsing of natural language specifications are developed, which perform parsing of the specification, determine the number and percentage of missing attributes and/or indicators, display which attributes and/or indicators are missing for the sub-characteristics of non-functional characteristics and/or metrics, and form a real ontology for non-functional characteristics and/or metrics, which can be used by OBIA for evaluating the initial stages of the software life cycle for assessing the sufficiency of information for determining the non-functional characteristics and metrics.

In addition, the developed OBIAs for semantic parsing the natural language SRS provide: automating and speeding up of the parsing the SRS; indicating the "bottlenecks" places (knowledge gaps) of SRS and demonstrating which requirements need re-work for providing the further assessment of non-functional characteristics-components of software quality; training for SRS developers and requirement engineers (they can see their mistakes and gaps in requirements); free online access, at any time, without any registration. A limitation of the developed OBIAs is the search for only the attributes defined by ISO 25023:2016 as necessary to calculate non-functional characteristics-components of software quality, as well as indicators defined by industry publications as necessary to calculate software complexity and quality metrics.

The ontology-based intelligent agents for evaluating the initial stages of the software life cycle are realized, which assess the sufficiency of information in the specification of requirements for determining all non-functional characteristics-components of software quality, as well as to calculate software metrics. Implemented intelligent agents provide a conclusion about the sufficiency or insufficiency of information in the specification. In addition, they provide numerical assessments of the level of sufficiency of information for determining each non-functional characteristic of the software and/or metrics and for determining all non-functional characteristics-components of the software quality together. Agents also form a list of attributes and indicators that should supplement the specification of requirements for improving the sufficiency of its information and visualize gaps in knowledge about all non-functional characteristics-components of software quality and software metrics. Therefore, the implemented intelligent agents provide automation of the analysis of the specifications of the software requirements for the sufficiency of their information. Thus, the presented agents allow to partially eliminate a person from the processes of information processing and knowledge acquisition.

All the results of the functioning of the realized ontology-based intelligent agents in the complex provide an increase of the level of sufficiency of information in the specification of software requirements for determining non-functional characteristics-components of software quality and software metrics. In addition, the results of the implementation of the realized intelligent agents are aimed at avoiding the loss of essential information and minimizing the occurrence of errors in the early stages of the software life cycle.

The agent-oriented information technology for assessing the sufficiency of information at the initial stages of the software life cycle was developed. This AOIT assesses and provides the increase (for example, from 58,23% till 86,02% for SRS No. 1, from 81,26% till 85,97% for SRS No. 1, from 60,85% till 73,7% for SRS No. 3) of the level of sufficiency of requirements information for determining software non-functional characteristics – the gain of the level of suffiiciency is from 4,71% to 27,79%.

In addition to the above, the advantages of the developed AOIT also are: 1) automation of the time-consuming, routine and error-prone task of parsing the SRS, and almost instantly accomplishment of it; 2) indication where re-work on SRS is needed (the user can browse missing attributes and see SRS areas which the extra attention is needed, and which requirements need re-work); 3) provision of training for new SRS developers, systems engineers and project managers (using this AOIT helps them see mistakes they might be making, and helps them recognize those mistakes in others' work); 4) help of developing the high-quality requirements; 5) help of correct and eliminate of requirements errors where they originate – during the early stages of the software project life cycle – before they become more expensive to correct; 6) provision of the tool for choosing the more qualitative software requirements specification; 7) free online access, at any time, without any registration.

The economic effect of the use of the developed AOIT is the ability to save software projects' budget for processing and correcting (during the life cycle) defects and bugs, which are made at the early stages of the life cycle – due to the demonstration of weaknesses in the SRS, that need to be finalized or re-worked, at a time when they arise.

The limitations of the developed AOIT are: 1) the assessment of the only sufficiency of requirements information for determining the non-functional characteristics as the sufficiency of the attributes in the SRS; 2) consideration of only non-functional characteristics, which are regimented by ISO 25010 standard as the software quality characteritics; 3) non-consideration the other SQUARE standards of 25000-series (ISO 25011, ISO 25012); 4) during parsing the SRS the search of only attributes, which are defined by ISO 25023 as necessary for the non-functional characteristics-components of software quality. For elimination of these limitations, further efforts of the authors will be directed.

Conclusions & Prospective Directions for Further Research

The monograph solves the current scientific and applied problem of developing the theoretical and applied principles of intelligent information and analytical technologies for improving the software quality by assessing the sufficiency of information at initial stages of the life cycle, which provide: conclusion on the sufficiency of quality information in the specification requirements; priority of supplementing the specification with the necessary information (in case of insufficient information) by forming a request to supplement the specification; evaluating and, if it's necessary, increasing the sufficiency of the quality information available in the specification; the ability to process quality information in the software requirements specifications by software agents (bots), without the participation of specialists, which provides the ability to automate such processes, eliminate the subjective influence of specialists and the safety of this information in the software company in case of specialist's dismissal.

The following scientific and practical results are obtained in the monograph:

1. A study of the current state of information technologies' development has shown that today the industry is undergoing significant changes due to the growing amount of information that needs to be processed and the imperfection of known methods and tools for processing large amounts of information. Today, the development of new information technologies the software engineering industry requires, especially in terms of software quality assurance. Studies of known models, methods and tools have shown that they do not solve the problem of assessing the sufficiency of quality information in the software requirements specifications. In addition, they all belong to different methodological approaches and are not integrated with each other, i.e. there are currently no intelligent information-analytical technologies for improving the quality of software by assessing the sufficiency of information at the early stages of the life cycle. The actuality of the problem of analyzing the sufficiency of information on quality in the specifications of software requirements necessitates the development of theoretical and applied principles for assessing the sufficiency of information on quality in the specifications of software requirements.

2. Theoretical bases of using ontologies for assessing the sufficiency of information on quality in the software requirements specifications and the criterion of the sufficiency of information on quality in the specifications of software requirements are developed, which are the basis for developing models of the process of assessing the sufficiency of information for determining the software quality based on ISO 25010:2011 and using the results of the metric analysis.

3. Models of the subject area "Software Engineering" (parts "Software Quality", "Software Quality. Metric Analysis") based on ontologies and weighted ontologies are developed, which provide the ability to display knowledge about the correlation of characteristics and sub-characteristics by attributes, metrics by indicators, and also the

possibility of modeling the assessment of the sufficiency of quality information in the requirements specifications.

4. Ontological models of the subject area "Software Engineering" (part "Specification of software requirements") are developed, which formalize the specification of software requirements in terms of the availability of quality information, and therefore are templates for developing a specification of software requirements in terms of software quality determination.

5. Models of the process of assessing the sufficiency of information for determining the quality of software based on ISO 25010:2011 and using the results of the metric analysis are developed, which, using comparative analysis of ontologies, provide a theoretical basis for developing methods and tools for assessing the sufficiency of information of specifications of software requirements, as well as provide the ability to assess the available information on quality in the specifications of requirements for the concrete software, to identify the content of insufficient information and to form the sequence of supplementing the specification with insufficient information.

6. Methods for assessing the sufficiency of quality information (according to ISO 25010:2011 and for metric analysis) in the specifications of software requirements based on ontologies are developed, which, based on comparative analysis of ontologies, allow to form a conclusion about the sufficiency of quality information in the requirements specification for the concrete software and the need to supplement the specification with attributes and (or) indicators.

7. Methods for assessing the sufficiency of quality information (according to ISO 25010:2011 and for metric analysis) in the specifications of software requirements based on weighted ontologies are developed, which, by marking the weights of attributes/indicators in base ontologies, make it possible to sort all missing in the specification requirements for software attributes and indicators in descending order of values of weights, i.e. to establish the priority of their supplementing to the specification of requirements for the software.

8. The developed methods for assessing the sufficiency of quality information (according to ISO 25010 and for metric analysis) in the software requirements specifications based on ontologies and weighted ontologies, due to their ability to supplement the requirements specification with the necessary information, make it possible to increase the sufficiency of quality information in the specification of software requirements, that reduces the size of the knowledge gap and the sector with unknown information about the software.

9. The monograph develops methods for generating and filling templates of ontology for determining the quality of the concrete software (based on ISO 25010 and metric information), which, leaving in the relevant ontology only attributes or indicators available in the specification of requirements for the concrete software, allow automated and easy to obtain an ontology for determining the quality of the concrete software according to ISO 25010 or based on metric information.

10. Methods of forming a logical conclusion about the sufficiency of information on quality (according to ISO 25010 and for metric analysis) in the specifications of software requirements are developed, which, based on the proposed production rules of forming a

logical conclusion about the sufficiency of information on quality, allow: to form a conclusion about sufficiency or insufficiency of quality information in the requirements specification; in case of insufficient information, to form conclusions for which sub-characteristics, characteristics and/or metrics information are insufficient, to form a sorted (by weights) list of missing attributes and/or indicators as a recommended priority of supplementing attributes and/or indicators in the specification; to assess the sufficiency of the amount of available (before and after the supplementing) quality information and to determine the gain of the sufficiency of the amount of quality information in the specification of requirements for the concrete software (after the supplementing).

11. In the monograph, the information technology for assessing the sufficiency of information on quality in the specifications of software requirements are developed, designed to support the assessment of software quality at the early stages of the life cycle. The developed information technology can be used for any classes of software for which the specification of software requirements is developed. Implemented information technology makes it possible to: 1) draw a conclusion about the sufficiency of information on quality in the specification of software requirements; 2) establish the priority of supplementing attributes and/or indicators in the specification of software requirements (in case of insufficient information in the specification) by forming a request to supplement software requirements; 3) evaluate and, if it's necessary, increase the sufficiency of the amount of quality information available in the specification of requirements; 4) process quality information in the software requirements specifications by software agents (bots), without the participation of specialists, which provides the ability to automate such processes and eliminate the subjective influence of specialists, as well as the safety of this information in the software company in case of specialist's dismissal. Thus, the developed information technology for assessing the sufficiency of quality information in the software requirements specifications is a new-generation information technology, which eliminates the person from the processes of processing quality information in the software requirements specification.

12. The structure and subsystem for assessing the sufficiency of information on quality in the requirements specifications based on the comparative analysis of ontologies are developed, designed for assessing the results of the stage of formation and formulation of requirements. The subsystem provides the user with conclusions about: the sufficiency of information on quality in the specification of requirements; the need to supplement the specification; the recommended priority of supplementing attributes and/or indicators in the specification, as well as a numerical assessment of the sufficiency of the amount of quality information available in the specification.

13. The practical significance of the obtained results is to develop information selection technology for the development of the base and weighted base ontologies of the subject area "Software Engineering" (parts "Software Quality", "Software Quality. Metric Analysis"), the base ontology of the subject area "Software Engineering" (parts "Specification of software requirements (quality attributes)", "Specification of software requirements (quality indicators)"), which provide a formalization of the subject area of software quality and specification of software requirements for access, analysis and

understanding of this information not only by man, but also virtual agents (bots); software requirements specification templates in terms of quality information (attributes and indicators); the possibility of developing a subsystem for assessing the sufficiency of information on quality in the specifications of software requirements. Also, the practical significance of the results is to develop information technology for assessing the sufficiency of quality information in the software requirements specifications, which provides: processing of quality information in the software requirements specifications by software agents (bots), without the participation of specialists; conclusion on the sufficiency of information on quality in the specification of requirements and the priority of supplementing the specification of requirements with the necessary information (in case of its insufficiency); assessing and increasing the sufficiency of the amount of information available in the specification of requirements; selecting a specification from a set of specifications in a situation where the cost and duration of software projects are approximately the same.

14. Studies have confirmed the practical value of the developed information technology for assessing the sufficiency of quality information in the software requirements specifications, which allowed to increase the sufficiency of quality information in the software requirements specification, as well as reduce the gap of knowledge about software quality assessment. The use of the developed information technology even after one re-work of the specification of requirements made it possible to increase the sufficiency of the amount of information on quality (attributes and) in the specification of requirements by 12% for a software agent for improving the safety of computer systems' software, as well as increase the sufficiency of information on quality (indicators) in the specification of requirements by 14% for a software agent for improving the safety of computer systems' software.

15. The monograph offers base (universal) ontological models of non-functional characteristics-components of software quality, as well as ontological models of non-functional characteristics-components of quality of the concrete software, which are based on the requirements of ISO 25010:2011 and ISO 25023:2016 and provide a basis for selection of sufficient amount of information for assessing the non-functional characteristics of the software. A model of activity of ontology-based intelligent agent for assessing the software requirements specifications is developed, which is based on comparative analysis of ontologies and is a theoretical basis for developing the methods of activity of the ontology-based intelligent agents.

16. Intelligent agents for semantic parsing the natural language specifications are developed, which perform parsing of the specification, determining the number and percentage of missing attributes and/or indicators, displaying which attributes and/or indicators are missing for a sub-characteristic of non-functional characteristics and/or metrics, and also form a real ontology for non-functional characteristics and/or metrics, which can be used by OBIA for evaluating the initial stages of the software life cycle for assessing the sufficiency of information for determining the non-functional characteristics and metrics.

17. The ontology-based intelligent agents for evaluating the initial stages of the software life cycle are developed, which assess the sufficiency of information in the

specification of requirements for determining all non-functional characteristics-components of software quality, as well as for calculating the software metrics. The realized intelligent agents provide a conclusion about the sufficiency or insufficiency of information in the specification. In addition, they provide a numerical assessment of the level of sufficiency of information for determining each non-functional characteristic of the software and/or metrics and for determining all non-functional characteristics-components of the quality of software and/or metrics together. Agents also form a list of attributes and indicators that should supplement the specification of requirements for improving the sufficiency of its information, and visualize gaps in knowledge about all non-functional characteristics-components of software quality and software metrics. Therefore, the realized intelligent agents provide automation of the analysis of the specifications of the software requirements for the sufficiency of their information. Thus, the presented agents allow to partially eliminate a person from the processes of information processing and knowledge acquisition.

18. The agent-oriented information technology for assessing the sufficiency of information at the initial stages of the software life cycle was developed. This AOIT assesses and provides the increase (for example, from 58,23% till 86,02% for SRS No. 1, from 81,26% till 85,97% for SRS No. 1, from 60,85% till 73,7% for SRS No. 3) of the level of sufficiency of requirements information for determining software non-functional characteristics – the gain of the level of sufficiency is from 4,71% to 27,79%. In addition to the above, the advantages of the developed AOIT also are: 1) automation of the time-consuming, routine and error-prone task of parsing the SRS, and almost instantly accomplishment of it; 2) indication where re-work on SRS is needed (the user can browse missing attributes and see SRS areas which the extra attention is needed, and which requirements need re-work); 3) provision of training for new SRS developers, systems engineers and project managers (using this AOIT helps them see mistakes they might be making, and helps them recognize those mistakes in others' work); 4) help of developing the high-quality requirements; 5) help of correct and eliminate of requirements errors where they originate – during the early stages of the software project life cycle – before they become more expensive to correct; 6) provision of the tool for choosing the more qualitative software requirements specification; 7) free online access, at any time, without any registration. The economic effect of the use of the developed AOIT is the ability to save software projects' budget for processing and correcting (during the life cycle) defects and bugs, which are made at the early stages of the life cycle – due to the demonstration of weaknesses in the SRS, that need to be finalized or re-worked, at a time when they arise.

The problem of software quality assurance today remains one of the main problems of software engineering. Early stages of the software life cycle, in particular the requirements formulation stage, are the least formalized and the most costly, as the costs of correcting incorrect specification requirements identified after product release are hundreds of times higher than the costs of correcting specification deficiencies identified during the requirements formulation process. The risks of an insufficiently worked out stage of formulating requirements are non-compliance with project deadlines and financial overspending, which can lead to the closure of the project or even the collapse of the software

company due to its financial instability.

One of the reasons for the low success of software systems is the lack of attention paid to information of the subject area at different stages of the life cycle, its sufficiency, veracity, accuracy. Information of the subject area can come at different stages of the life cycle – both at the initial stages and at the stages of implementation and operation. Some information of the subject area is analyzed meticulously, but some information of the subject area is not analyzed at all. Developers often neglect subject area information with a low probability, and sometimes they neglect it without even estimating its probability. Such neglect of subject area information at all stages of the life cycle is one of the critical factors in software system development.

The development of software systems requires taking into account the information of the subject area at all stages of its design and development in order to increase its reliability, functional safety, survivability, resiliency and dependability. Therefore, verification of taking into account the information of the subject area in the process of software development is an important and actual task. For solving this task, it's necessary to develop an intelligent agent, which will verify the consideration of information of the subject area in the process of software development.

In the monograph, the authors developed a method of activity of ontology-based intelligent agent for evaluating the initial stages of the software life cycle. This method is based on a comparison of ideal and real ontologies, which contain information on quality in the specification of software requirements, and the ideal ontology contains all the necessary information on quality and the real ontology contains available information on quality. Therefore, a comparison of these two ontologies will make it possible to see the losses of information (and knowledge gaps) on quality in the specification of software requirements, as well as to estimate their amount.

This approach can be used to describe *the activity of the intelligent agent for verifying the consideration of the subject area's information in the entire life cycle of software systems*.

Regardless of the chosen model of the life cycle, at each stage of the software life cycle, a certain document is formed (plan, specification of software requirements, design (architecture) of the software system, source code, product, etc.), which must have a certain (usually defined by the relevant standard) information.

Then, given what information should be available in the relevant document, it is necessary to develop ideal ontologies – for each document. At the same time, the main attention should be paid to the information of the subject area, which can and should be taken into account in the relevant document at the appropriate stage of the software life cycle.

In the process of working on a real software system, developers will form such real documents, then on their basis, it is necessary to form real ontologies – for each document.

After that, it is necessary to compare the real ontologies developed by each corresponding real document (a stage of the software life cycle), with the corresponding ideal ontologies developed earlier. A comparison of the two relevant ontologies for each document will provide a set of information elements that are missing in the relevant documents, which will allow identifying the losses of information of the subject area in this document, as well

as assessing their amount. For assessing the amount of information loss, the proposed in the monograph formulas for numerical assessment of the level of sufficiency of available information can be used – with adaptation to the relevant document and the information available in it.

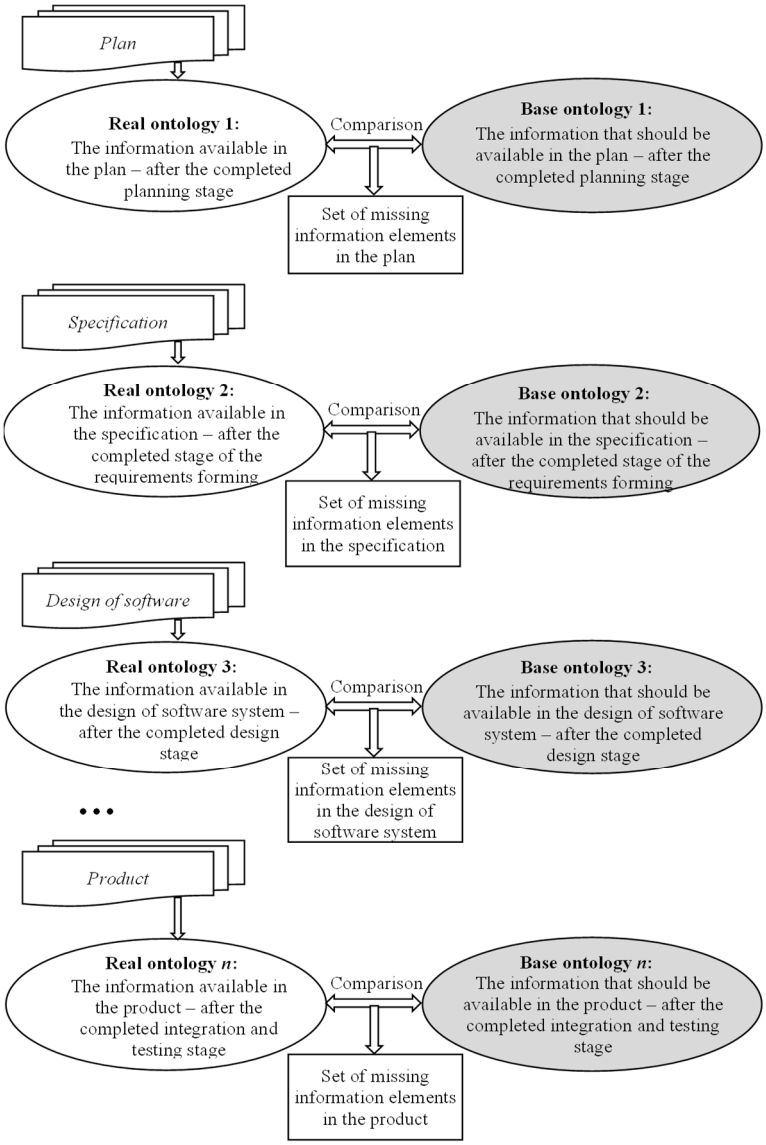

Fig. C.1. Concept of activity of intelligent agent of verification of considering the subject area's information in the process of software systems' development [155]

Concept of activity of intelligent agent of verification of considering the subject area's information in the process of software systems' development is presented in Fig. C.1 [155].

Such an intelligent agent, after processing the sets of missing (at each life cycle stage) information elements using certain production rules, will make conclusions about the sufficiency/insufficiency of information at each stage of the life cycle, the level of considering the subject area's information at each stage of the life cycle, and will provide the visual recommendations for what information should be consider for increasing its sufficiency at each stage of the software systems' development life cycle.

Therefore, the prospective directions of further research of authors are: development of base ontologies for all documents at each stage of the software systems' life cycle – on the basis of relevant standards, guidelines, etc.; modeling and developing the method of activity of the intelligent agent of verification of considering the subject area's information in the process of software systems' development; realization of the intelligent agent of verification of considering the subject area's information in the process of software systems' development.

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:

$$SRS=<S\_1,...,S\_19>,$$

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:

$$S\_1 = \{ps\},$$

$$S\_2 = \{isp, spd, spb, spo, spg, cssh\},$$

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

$$S\_4 = \{mf\},$$

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

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

S_7 = {far},

S_8 = {asrse, crtf, crte, rduf},

S_9 = {rlds, iss, oss, fss},

S_10 = {ni, dp, sido, vrat, um, tg, roio, sfo, wfo, df, cmf, emsg},

S_11 = {faapi, fapgo, vci, eso, ras, ep, rsoi},

S_12 = {ubr, mec, mefc, msc},

S_13 = {snr, dnr, nts, nssu, athi, nota, not, adnw, adpw, prq},

S_14 = {tiuvf, fqu, asc, der, ics, drr},

S_15 = {ces, crr, cpl},

S_16 = {rf, dn, ap, at},

S_17 = {rb, avb, scr, cct, slhd, fdm, csa, dicv, dp,

mb, pb, pehdc, pchd, uppl, upcls, upos},

S_18 = {va, mqs},

S_19 = {siof, dcas, rus, sbi, dpss, spi},

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.

The above equations 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 above equations, 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.

Approach to the analysis of software requirements specification on its structure correctness (according to ISO/IEC/IEEE 29148:2018) is represented in Fig. C.2.

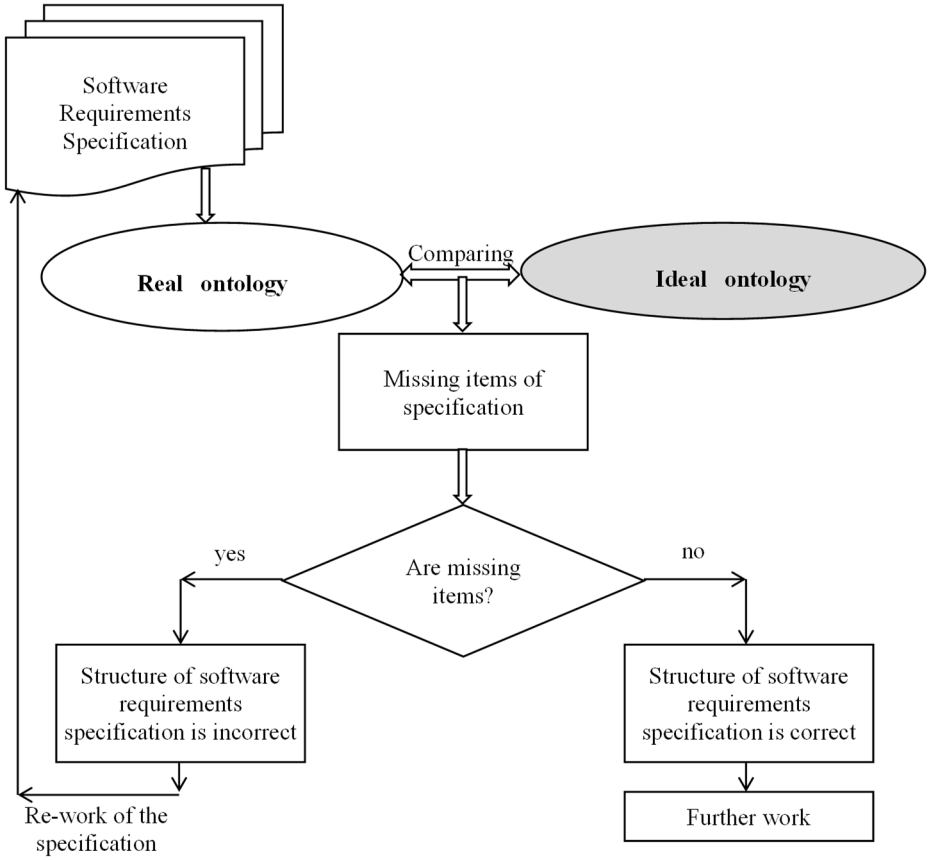

Fig. C.2. Approach to the analysis of software requirements specification on its structure correctness (according to ISO/IEC/IEEE 29148:2018) [156]

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.

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 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).

Bibliography

[1] SIVARAJAH, U., KAMAL, M., IRANI, Z., WEERAKKODY, V.: *Critical analysis of Big Data challenges and analytical methods*, Journal of Business Research, 2017, vol. 70, pp. 263-286.

[2] LUO, Y. D., BU, J.: *How valuable is information and communication technology? A study of emerging economy enterprises*, Journal of World Business, 2016, vol. 51, issue 2, pp. 200-211.

[3] TIAN, X.M.: *Big data and knowledge management: a case of deja vu or back to the future?*, Journal of Knowledge Managament, 2017, vol. 21, issue 1, pp. 113-131.

[4] MAUERHOEFER, T., STRESE, S., BRETTEL, M.: *The impact of information technology on new product development performance*, Journal of Product Innovation Management, 2017, vol. 34, issue 6, pp. 719-738.

[5] QIU, J., WU, Q., DING, G., XU, Y., FENG, S.: *A survey of machine learning for big data processing*, EURASIP Journal on Advances in Signal Processing, 2016, URL: https://www.springeropen.com/track/pdf/10.1186/s13634-016-0355-x?site=asp-eurasipjournals.springeropen.com (Last accessed: May 20, 2020).

[6] KINCH, M. W., MELIS, W. J. C., KEATES, S.: *Reviewing the current state of machine learning for artificial intelligence with regards to the use of contextual information*, The Second Medway Engineering Conference on Systems: Efficiency, Sustainability and Modeling: Proceedings (Greenwich, UK, June 6, 2017).

[7] NOOR, A. K.: *Potential of cognitive computing and cognitive systems*, Open Engineering, 2015, vol. 5, issue 1, pp. 75-88.

[8] ABBASI, A., SARKER, S., CHIANG, R. H. L.: *Big data research in information systems: toward an inclusive research agenda*, Journal of the Association for Information Systems, 2016, vol. 17, issue 2, article 3.

[9] LECUN, Y., BENGIO, Y., HINTON, G.: *Deep learning*, Nature, 2015, vol. 521, issue 7553, pp. 436-444.

[10] GHEISARI, M., WANG, G., BHUIYAN, M. Z. A.: *A Survey on deep learning in big data*, 2017 IEEE International Conference on Computational Science and Engineering and Embedded and Ubiquitous Computing: Proceedings (Guangzhou, China, July 21-24, 2017), pp. 173-180.

[11] RISTOSKI, P., PAULHEIM, H.: *Semantic Web in data mining and knowledge discovery: A comprehensive survey*, Journal of WEB Semantics, 2016, vol. 36, pp. 1-22.

[12] OSTROWSKI, D., RYCHTYCKYJ, N., MACNEILLE, P., KIM, M.: *Integration of big data using semantic web technologies*, 2016 IEEE Tenth International Conference

on Semantic Computing: Proceedings (Laguna Hills, USA, March 24, 2016), pp. 382-385.

[13] PRESS, G. *Top 10 hot artificial intelligence (AI) technologies*, Web-site, URL: https://www.forbes.com/sites/gilpress/2017/01/23/top-10-hot-artificial-intelligence-ai-technologies/#1d5da9561928 (Last accessed: May 20, 2020).

[14] GOLUB, K.: *Subject access to information: An interdisciplinary approach*, Libraries Unlimited, 2015, ISBN: 978-1610695770.

[15] MCCONNELL, S.: *Code complete*, Microsoft Press, 2013, ISBN: 978-0735619678

[16] MAEDCHE, A., BOTZENHARDT, A., NEER, L.: *Software for people: fundamentals, trends and best practices (Management for professionals)*, Springer-Verlag Berlin Heidelberg, 2012, ISBN: 978-3642313707.

[17] *Latest study shows rise in project failures*, Web-site, URL: http://kinzz.com/resources/articles/91-project-failures-rise-study-shows (Last accessed: May 20, 2020).

[18] *CHAOS Summary 2009. The 10 laws of CHAOS*, Web-site, URL: https://www.classes.cs.uchicago.edu/archive/2014/fall/51210~1/required.reading/Standish.Group.Chaos.2009.pdf (Last accessed: May 20, 2020).

[19] *The Standish Group International: CHAOS Manifesto – Think big, act small. Technical report 2013*, Web-site, URL: http://www.versionone.com/assets/img/files/CHAOSManifesto2013.pdf (Last accessed: May 20, 2020).

[20] SHANE, H., STÉPHANE W.: *Standish Group 2015 Chaos Report – Q&A with Jennifer Lynch*, Web-site, URL: http://www.infoq.com/articles/standish-chaos-2015 (Last accessed: May 20, 2020).

[21] *The Standish Group Report CHAOS*, Web-site, URL: https://www.projectsmart.co.uk/white-papers/chaos-report.pdf (Last accessed: May 20, 2020).

[22] *PMI's Pulse of the Profession 9-th Global Project Management Surve*y, 2017, Web-site, URL: https://www.pmi.org/-/media/pmi/documents/public/pdf/learning/thought-leadership/pulse/pulse-of-the-profession-2017.pdf (Last accessed: May 20, 2020).

[23] LEONARD, J.: *CSCE 606: Introduction*, Web-site, URL: https://slideplayer.com/slide/15545897/ (Last accessed: May 20, 2020).

[24] MERSINO, A.: *Agile Project Success Rates are 2X Higher than Traditional Projects*, Web-site, URL: https://vitalitychicago.com/blog/agile-projects-are-more-successful-traditional-projects/ (Last accessed: May 20, 2020).

[25] *4-Step Project Plan for 2020*, Web-site, URL: http://projexs.io/project-plan-made-easy/ (Last accessed: May 20, 2020).

[26] *A Look at 25 Years of Software Projects. What Can We Learn?*, Web-site, URL: https://speedandfunction.com/look-25-years-software-projects-can-learn/ (Last accessed: May 20, 2020).

[27] LEVENSON, N. G.: *Engineering a safer world: systems thinking applied to safety*, MIT Press, 2012, ISBN: 978-0262533690.

[28] *Cost of a bug within a software life cycle*, Web-site, URL: http://www.testically.org/2012/02/09/cost-of-a-bug-within-a-software-life cycle/ (Last accessed: May 20, 2020).

[29] *Leveraging Natural Language Processing in Requirements Analysis: How to eliminate over half of all design errors before they occur*, Web-site, URL: http://qracorp.com/wp-content/uploads/2017/03/Leveraging-NLP-in-Requirements-Analysis.pdf (Last accessed: May 20, 2020).

[30] SHAMIEH, C.: *Systems Engineering for Dummies*, Wiley Publishing, 2014, ISBN: 978-1118100011.

[31] ISO/IEC 25010:2011. Systems and software engineering. Systems and software Quality Requirements and Evaluation (SQuaRE). System and software quality models. [Introduced 01.03.2011]. Geneva (Switzerland), 2011. 34 p. (International standard).

[32] ISO 25023:2016. Systems and software engineering. Systems and software Quality Requirements and Evaluation (SQuaRE). Measurement of system and software product quality. [Introduced 31.03.2016]. Geneva (Switzerland), 2016. 45 p. (International standard).

[33] KHARCHENKO, A., GALAY, I., YATCYSHYN, V.: *The method of quality management software*, VII International Conference on Perspective Technologies and Methods in MEMS Design: Proceedings (Polyana, May 11-14, 2011), pp. 82-84.

[34] SOMMERVILLE, I.: *Software Engineering*, Pearson, 2017, ISBN: 978-9332582699.

[35] JONES, C., BONSIGNOUR, O.: *The economics of software quality*, Pearson Education, 2012, ISBN: 978-0132582209.

[36] WIEGERS, K., BEATTY, J.: *Software requirements: 3rd edition*, MS Press, 2013, ISBN: 978-0735679665.

[37] FENTON, N., BIEMAN, J.: *Software metrics: a rigorous and practical approach*, CRC Press, 2014, ISBN: 978-1439838228.

[38] CHEN, A., BEATTY, J.: *Visual models for software requirements*, MS Press, 2012, ISBN: 978-0735667723.

[39] FATWANTO, A.: *Software requirements specification analysis using natural language processing technique*, The 13-th IEEE International Conference on Quality in Research: Proceedings (Yogyakarta, Indonesia, June 25-28, 2013), pp. 105-110.

[40] REHMAN, T., KHAN, M. N. A., RIAZ, N.: *Analysis of requirement engineering processes, tools/techniques and methodologies*, International Journal of Information Technology and Computer Science, 2013, vol. 5, no. 3, pp. 40-48.

[41] BUROV, E.: *Complex ontology management using task models*, International Journal of Knowledge-Based and Intelligent Engineering Systems, 2014, vol. 18, no. 2, pp. 111-120.

[42] LI, Y., CLELAND-HUANG, J.: *Ontology-based trace retrieval*, The 7-th International Workshop on Traceability in Emerging Forms of Software Engineering: Proceedings (Passau, Germany, May 19, 2013), pp. 30-36.

[43] ASSAWAMEKIN, N., NAMFON, A., SUNETNANTA, T., PLUEMPITIWIRIYAWEJ, C.: *Ontology-based multiperspective requirements traceability framework*, Knowledge Information Systems, 2010, no. 3, pp. 493-522.

[44] HENDERSON-SELLERS, B., GONZALEZ-PEREZ, C., MCBRIDE, T., LOW, G.: *An ontology for ISO software engineering standards: 1) Creating the infrastructure*, Computer Standards & Interfaces, 2014, vol. 36, issue 3, pp. 563-576.

[45] RUY, F., FALBO, R., BARCELLOS, M., GUIZZARDI, G., QUIRINO, G.: *An ISO-based software process ontology pattern language and its application for harmonizing standards*, ACM SIGAPP Applied Computing Review, 2015, vol. 15, issue 2, pp. 27-40.

[46] HAMRI, M. M., BENSLIMANE, S. M.: *Building an ontology for the metamodel ISO/IEC24744 using MDA process*, International Journal on Modern Education and Computer Science, 2015, vol. 8, pp. 48-60.

[47] BAJNAID, N. O., BENLAMRI, R., PAKSTAS, A., SALEKZAMANKHANI, S.: *An ontological approach to model software quality assurance knowledge domain*, Lecture Notes on Software Engineering, 2016, vol. 4, no. 3, pp. 193-198.

[48] ISO/IEC 25030:2019. Systems and software engineering. Systems and software quality requirements and evaluation (SQuaRE). Quality requirements framework [Introduced 01.09.2019]. Geneva (Switzerland), 2019. 46 p. (International standard).

[49] ISO/IEC TR 19759:2015. Software Engineering. Guide to the software engineering body of knowledge (SWEBOK). [Introduced 01.10.2015]. Geneva (Switzerland), 2015. 336 p. (International standard).

[50] ISO 9000:2015. Quality management systems. Fundamentals and vocabulary. [Introduced 15.09.2015]. Geneva (Switzerland), 2015. 51 p. (International standard).

[51] ISO 9001:2015. Quality management systems. Requirements. [Introduced 15.09.2015]. Geneva (Switzerland), 2015. 29 p. (International standard).

[52] ISO 10002:2018. Quality management. Customer satisfaction. Guidelines for complaints handling in organizations. [Introduced 01.08.2018]. Geneva (Switzerland), 2018. 31 p. (International standard).

[53] HOVORUSHCHENKO, T.: *Software quality evaluation and assurance: analysis, tendencies, problems*, The 6-th International Conference on Advanced Computer Systems and Networks: Design and Application: Proceedings (Lviv, September 16-18, 2013), pp. 142-145.

[54] POMOROVA, O., HOVORUSHCHENKO, T.: *Intelligent Assessment and Prediction of Software Characteristics at the Design Stage*, American Journal of Software Engineering and Applications, 2013, vol. 2, issue 2, pp. 25-31.

[55] HOVORUSHCHENKO, T., KRASIY, A.: *Method of Evaluating the Success of Software Project Implementation Based on Analysis of Specification Using Neuronet Information Technologies*, CEUR-WS, 2015, vol. 1356, pp. 100-107.

[56] POMOROVA, O., HOVORUSHCHENKO, T.: *The Way to Detection of Software Emergent Properties*, The 8-th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications: Proceedings (Warsaw, Poland, September 24-26, 2015), vol. 2, pp. 779-784.

[57] HOVORUSHCHENKO, T., KRASIY, A.: *Realization of Neural Network Model of Prediction of the Software Project Characteristics for Evaluating the Success of Its Implementation*, The 8-th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications: Proceedings (Warsaw, Poland, September 24-26, 2015), vol.1, pp. 348-353.

[58] KRASIY, A., HOVORUSHCHENKO, T.*: Information Technology of Predicting the Characteristics and Evaluating the Success of Software Projects Implementation*, CEUR-WS, 2016, vol.1614, pp.87-102.

[59] *Delivering large-scale IT projects on time, on budget, and on value*, Web-site, URL: https://www.mckinsey.com/~/media/McKinsey/dotcom/client_service/BTO/PDF/MOBT_27_Delivering_large-scale_IT_projects_on_time_budget_and_value.ashx (Last accessed: May 21, 2020).

[60] DOROFEE, A., WOODY, C., ALBERTS, C., CREEL, R., ELLISON, R.: *A Systemic approach for assessing software supply-chain risk*, The 44-th Hawaii International Conference on System Sciences: Proceedings (Honolulu, USA, January 04-07, 2011), pp. 1-8.

[61] GHADGE, A., DANI, S., CHESTER, M., KALAWSKY, R.: *A systems approach for modeling supply chain risks*, Supply Chain Management: An International Journal, 2013, vol. 18 (5), pp. 523-538.

[62] HOVORUSHCHENKO, T.: *The Software Emergent Properties and them Reflection in the Non-Functional Requirements and Quality Models*, 10-th International Conference on Computer Science and Information Technologies: Proceedings (Lviv, September 14-17, 2015), pp. 146-153.

[63] LYNCH, J.: *Project Resolution Benchmark Report*, Web-site, URL: https://www.standishgroup.com/sample_research_files/DemoPRBR.pdf (Last accessed: May 21, 2020).

[64] JOHNSON, J.: *CHAOS Report: Decision Latency Theory: It Is All About the Interval*, The Standish Group, 2018, ISBN: 978-0692048306.

[65] SARIF, S., RAMLY, S., YUSOF, R., FADZILLAH, N. A. A., SULAIMAN, N. Y.: *Investigation of Success and Failure Factors in IT Project Management*, Advances in Intelligent Systems and Computing, 2018, vol. 739, pp. 671-682.

[66] ROSATO, M.: *Go Small for Project Success*, PM World Journal, 2018, vol. 7, issue 5, pp. 1-10.

[67] MEZIANE, F., VADERA, S.: *Artificial Intelligence Applications for Improved Software Engineering Development: New Prospects*, Advances in Intelligent Information Technologies, 2010, pp. 278-299.

[68] MAIER, R.: *Knowledge management systems. Information and communication technologies for knowledge management*, Springer Science & Business Media, 2013, ISBN: 978-3540714088.

[69] ISHIMATSU, T., LEVENSON, N., THOMAS, J., FLEMING, C., KATAHIRA, M., MIYAMOTO, Y., UJIIE, R., NAKAO, H., HOSHINO, N.: *Hazard analysis of complex spacecraft using systems-theoretic process analysis*, Journal of Spacecraft and Rockets, 2014, vol. 51, no. 2, pp. 509-522.

[70] MUNCH, J., SCHMID, K.: *Perspectives on the future of software engineering*, Springer-Verlag Berlin Heidelberg, 2013, ISBN: 978-3642373947.

[71] DIAZ, V. G., LOVELLE, J. M., GARCIA-BUSTELO, B. C.: *Handbook of research on innovations in systems and software engineering*, Hershey, 2015, ISBN: 978-1466663596.

[72] MISHRA, J., MOHANTY, A.: *Software engineering*, Pearson, 2012, ASIN: B00BIZS35K.

[73] PATTERSON, J. F. G.: *Life cycles for system acquisition,* Encyclopedia of Life Support Systems, Systems Engineering and Management for Sustainable Development, 2004, pp. 82-110.

[74] HOVORUSHCHENKO, T.: *Methodology of Evaluating the Sufficiency of Information for Software Quality Assessment According to ISO 25010*, Journal of Information and Organizational Sciences, 2018, vol. 42, no.1, pp. 63-85.

[75] HOVORUSHCHENKO, T., POMOROVA, O.: *Information Technology of Evaluating the Sufficiency of Information on Quality in the Software Requirements Specifications*, CEUR-WS, 2018, vol.2104, pp.555-570

[76] HOVORUSHCHENKO, T., POMOROVA, O.: *Methodology of Evaluating the Sufficiency of Information on Quality in the Software Requirements Specifications,*

2018 IEEE 9th International Conference on Dependable Systems, Services and Technologies: Proceedings (Kyiv, May 24-27, 2018), pp. 385-389.

[77] HOVORUSHCHENKO, T., BOYARCHUK, A., PAVLOVA, O., BOBROVNIKOVA, K.: *Agent-Oriented Information Technology for Assessing the Initial Stages of the Software Life Cycle*, CEUR-WS, 2019, vol. 2393, pp.617-632.

[78] CRUICKSHANK, K. J.: *A validation metrics framework for safety-critical software-intensive systems*. Naval Postraduate School, 2009.

[79] 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, 2010, vol. 4, issue 2, pp. 186-197.

[80] BAKER, R., HABLI, I.: *An empirical evaluation of mutation testing for improving the test quality of safety-critical software*, IEEE Transactions on Software Engineering, 2013, vol. 39, issue 6, pp. 787-805.

[81] *Software Requirements Engineering Tools*, Web-site, URL: http://ecomputernotes.com/software-engineering/softwarerequirements engineeringtools (Last accessed: May 21, 2020).

[82] *30 Best Requirements Management Tools in 2019*, Web-site, URL: https://www.guru99.com/requirement-management-tools.html (Last accessed: May 21, 2020).

[83] *Top 20+ Best Requirements Management Tools (The Complete List)*, Web-site, URL: https://www.softwaretestinghelp.com/requirements-management-tools/ (Last accessed: May 21, 2020).

[84] VERMA, K., KASS, A.: *Requirements Analysis Tool: A Tool for Automatically Analyzing Software Requirements Documents*, Lecture Notes in Computer Science, 2008, vol. 5318, pp. 751-763.

[85] MAHMOOD, H., REHMAN, M. S.: *Tools for software engineers*, International Journal of Research in Engineering & Technology, 2015, vol. 3, issue 10, pp. 75-86.

[86] JONES, V., MURRAY, J.: *Evaluation of current requirements analysis tools capabilities for IV&V in the requirements analysis phase*, Web-site, URL: https://www.slideserve.com/shlomo/evaluation-of-current-requirements-analysis-tools-capabilities-for-ivv-in-the-requirements-analysis-phase (Last accessed: May 21, 2020).

[87] RAFFO, D. M., FERGUSON, R.: *Evaluating the Impact of The QuARS Requirements Analysis Tool Using Simulation*, Web-site, URL: https://pdfs.semanticscholar.org/7e7d/4e6f5ab13d00ca57c87711e30cd080730f34.pdf (Last accessed: May 21, 2020).

[88] KONIG, F., BALLEJOS, L., ALE, M.: *A semi-automatic verification tool for software requirements specification documents*, Simposio Argentino de Ingeniería de Software: Proceedings (Cordoba, September, 2017), pp.75-83.

[89] ABRAN, A., AL-QUITASH, R. E., DESHARNAIS, J.-M., HABRA, N.: *ISO-based models to measure software product quality*, Software Quality Measurement: Concepts and Approaches, 2014, chapter 5, pp. 61-96.

[90] MONTAGUD, S., ABRAHAO, S., INSFRÁN, E.: *A systematic review of quality attributes and measures for software product lines*, Software Quality Journal, 2012, No. 20 (3-4), pp. 425-486.

[91] HER, J. S., KIM, J. H., OH, S. H., RHEW, S. Y., KIM, S. D.: *A framework for evaluating reusability of core asset in product line engineering*, Information and Software Technology, 2007, no. 49, pp. 740-760.

[92] BISCOGLIO, I., MARCHETTI, E.: *Definition of software quality evaluation and measurement plans: a reported experience inside the audio-visual preservation context*, The 9-th International Joint Conference on Software Technologies: Proceedings (Vienna, Austria, August 29-31, 2014), pp. 63-80.

[93] HOVORUSHCHENKO, T.: *Method of Evaluating the Weights of Software Quality Measures and Indicators*, Application and Theory of Computer Technology, 2017, vol.2, no.2, issue 2, pp.16-25.

[94] HOVORUSHCHENKO, T., POMOROVA, O.: *Evaluation of Mutual Influences of Software Quality Characteristics Based ISO 25010:2011*, 11-th International Conference on Computer Science and Information Technologies: Proceedings (Lviv, September 06-10, 2016), pp. 80-83.

[95] SUGIYANTO, S., ROCHIMAN, S.: *Integration of DEMATEL and ANP methods for calculate the weight of characteristics software quality based model ISO 9126*, The 5-th International Conference on Information Technology and Electrical Engineering: Proceedings (Yogyakarta, Indonesia, October 7-8, 2013), pp.143-148.

[96] ISO/IEC/IEEE 24765:2017. Systems and software engineering. Vocabulary. [Introduced 01.10.2017]. Geneva (Switzerland), 2017. 522 p. (International standard).

[97] POMOROVA, O., HOVORUSHCHENKO, T.: *The Intelligent Decision Support System for Choice of Software Project*, Journal of Information, Control and Management Systems, 2012, vol. 10, no. 1, pp.87-96.

[98] POMOROVA, O., HOVORUSHCHENKO, T.: *Research of Artificial Neural Network's Component of Software Quality Evaluation and Prediction Method*, The 6-th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications: Proceedings (Prague (Czech Republic), September 15-17, 2011), vol. 2, pp. 959-962.

[99] POMOROVA, O., HOVORUSHCHENKO, T.: *Artificial Neural Network for Software Quality Evaluation Based on the Metric Analysis*, IEEE East-West Design & Test Symposium: Proceedings (Kharkiv, September 14-17, 2012), pp. 200-203.

[100] KAN, S.: Metrics and models in software quality engineering, Addison-Wesley Professional, 2002, ISBN: 978-0201729153.

[101] WU, W., LI, H., WANG, H., ZHU, K. Q.: *Probase: a probabilistic taxonomy for text understanding*, The 2012 ACM International Conference on Management of Data (SIGMOD): Proceedings (Scottsdale, USA, May 20-24, 2012), pp.13-24.

[102] ROUSSEY, C., PINET, F., KANG, M. A., CORCHO, O.: *An Introduction to Ontologies and Ontology Engineering. Ontologies in Urban Development Projects*, Advanced Information and Knowledge Processing, 2011, vol.1, pp. 9-38.

[103] BUROV, E., PASITCHNYK, V., GRITSYK, V.: *Modeling software testing processes with task ontologies*, British Journal of Education and Science, 2014, no. 2(6), pp. 256-263.

[104] ZHANG, Y. G., WITTE, R., RILLING, J., HAARSLEV, V. *Ontological approach for the semantic recovery of traceability links between software artefacts*, IET Software, 2008, vol. 2, issue 3, pp. 185-203.

[105] LEONID, K., GACITUA, R., ROUNCEFIELD, M., SAWYER, P.: *Ontology and model alignment as a means for requirements validation*, The 4-th IEEE International Conference on Semantic Computing: Proceedings (Pittsburgh, USA, September 22-24, 2010), pp. 46-51.

[106] JIN, D., CORDY, J. R.: *Ontology-based software analysis and reengineering tool integration: The OASIS service-sharing methodology*, The 21-st IEEE International Conference on Software Maintenance: Proceedings (Budapest, Hungary, September 25-30, 2005), pp. 613-616.

[107] WOOLDRIDGE, M., JENNINGS, N. R.: *Intelligent agents – theory and practice*, Knowledge Engineering Review, 1995, vol. 10, issue 2, pp. 115-152.

[108] WEISS, G.: *Multiagent systems*, The MIT Press. 2013, ISBN: 978-0262232036.

[109] 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, 2017, vol. 264, pp. 40-55.

[110] LYTVYN, V., OBORSKA, O., VOVNJANKA, R.: *Approach to decision support intelligent systems development based on ontologies*, Econtechmod, 2015, vol. 4, no 4, pp. 29-35.

[111] FREITAS, A., PANISSON, A. R., HILGERT, L., MENEGUZZI, F., VIEIRA, R., BORDINI, R. H.: *Applying ontologies to the development and execution of Multi-Agent Systems*, Web Inteliigence, 2017, vol. 15, issue 4, pp. 291-302.

[112] FREITAS, A., BORDINI, R. H., VIEIRA, R.: *Model-driven engineering of multi-agent systems based on ontologies*, Applied Ontology, 2017, vol. 12, issue 2, pp. 157-188.

[113] LEZCANO-RODRIGUEZ, L. A., GUZMAN-LUNA, J. A.: *Ontological characterization of basics of KAOS chart from natural language*, Revista Iteckne, 2016, vol. 13, issue 2, pp. 157-168.

[114] HILAIRE, V., COSSENTINO, M., GECHTER, F., RODRIGUEZ, S., KOUKAM, A.: *An approach for the integration of swarm intelligence in MAS: An engineering perspective*, Expert Systems with Applications, 2013, vol. 40, issue 4, pp. 1323-1332.

[115] GARCIA-MAGARINO, I., GOMEZ-SANZ, J. J.: *An ontological and agent-oriented modeling approach for the specification of intelligent Ambient Assisted Living systems for Parkinson patients*, Hybrid Artificial Intelligent Systems, 2013, vol. 8073, pp. 11-20.

[116] WILK, S., MICHALOWSKI, W., O'SULLIVAN, D., FARION, K., SAYYAD-SHIRABAD, J., KUZIEMSKY, C., KUKAWKA, B.: *A task-based support architecture for developing point-of-care clinical decision support systems for the emergency department*, Methods of Information in Medicine, 2013, vol. 52, issue 1, pp. 18-32.

[117] RAKIB, A., FARUQUI, R. U.: *A formal approach to modeling and verifying resource-bounded context-aware agents*, Lecture Notes of the Institute for Computer Sciences Social Informatics and Telecommunications Engineering, 2013, vol. 109, pp. 86-96.

[118] MEYER, O., GRUHN, V.: *Towards Concept based Software Engineering for Intelligent Agents*, 2019 IEEE/ACM 7-th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE): Proceedings, 2019.

[119] DE OLVEIRA, C. D. C., CINTRA, M. E., NETO, F. M. M.: *Learning Risk Management in Software Projects with a Serious Game Based on Intelligent Agents and Fuzzy Systems*, The 8-th Conference of the European Society for Fuzzy Logic and Technology (EUSFLAT): Proceedings, 2013.

[120] YAN, H.: *Related Discussion on Agent-oriented Programming*, AER-Advances in Engineering Research, 2016, vol. 67, pp. 1315-1317.

[121] RAHMAN, A. A., ABDULLAH, M., ALIAS, S. H.: *The Architecture of Agent-Based Intelligent Tutoring System for the Learning of Software Engineering Function Point Metrics*, 2016 2-nd International Symposium on Agent, Multi-Agent Systems and Robotics (ISAMSR): Proceedings, 2016.

[122] SIMONS, C. L., PARMEE, I. C.: *User-centered, Evolutionary Search in Conceptual Software Design*, IEEE Congress on Evolutionary Computation, 2008, pp. 869-876.

[123] AKKAWI, F., BADER, A., ELRAD, T.: *The multi-layered approach to building intelligent systems*, International Conference on Artificial Intelligence: Proceedings, 2001.

[124] KARASEV, V. O., SUKHANOV, V. A.: *Product life cycle management using multi-agent systems models*, Procedia Computer Science, 2017, vol. 103, pp. 142-147.

[125] ABADI, A., SEKKAT, S., ZEMMOURI, E., BENAZZA, H.: *Using ontologies for the integration of information systems dedicated to product (CFAO, PLM...) and those of systems monitoring (ERP, MES...)*, 10-th International Colloquium on Logistics and Supply Chain Management (LOGISTIQUA): Proceedings, 2017.

[126] PHILIP, T., KONDA, R.: *A model to use software agents during software life-cycle*, Computers and Their Applications, 2001, pp. 530-536.

[127] PAVLOVA, O., HOVORUSHCHENKO, T., BOYARCHUK, A.: *Method of activity of intelligent agent for semantic analysis of software requirements*, The 2019 IEEE 10-th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS-2019): Proceedings, 2019, vol.2, pp. 902-906.

[128] HOVORUSHCHENKO, T., BOYARCHUK, A., PAVLOVA, O.: *Ontology-Based Intelligent Agent for Semantic Parsing the Software Requirements Specifications*, International Journal on Information Technologies and Security, 2019, no. 2, vol. 11, pp.59-70.

[129] GULIA, S., CHOUDHURY, T.: *An Efficient Automated Design to Generate UML Diagram From Natural Language Specifications*, The 6-th International Conference on Cloud System and Big Data Engineering: Proceedings, 2016, pp. 641-648.

[130] SELWAY, M., GROSSMAN, G., MAYER, W., STUMPTNER, M.: *Formalising natural language specifications using a cognitive linguistic/configuration based approach*, Information Systems, 2015, vol. 54, pp. 191-208.

[131] ALI, S. W., AHMED, Q. A., SHAFI, I.: *Process to Enhance the Quality of Software Requirement Specification Document,* The International Conference on Engineering and Emerging Technologies: Proceedings, 2018, pp. 113-118.

[132] DIAMANTOPOULOS, T., ROTH, M., SYMEONIDIS, A., KLEIN, E.: *Software requirements as an application domain for natural language processing*, Language Resources and Evaluation, 2017, vol. 51 (2), pp. 495-524.

[133] WANG, Y.: *Semantic Information Extraction for Software Requirements using Semantic Role Labeling*, The IEEE International Conference on Progress in Informatics and Computing: Proceedings, 2015, pp. 332-337.

[134] RIAZ, M., KING, J., SLANKAS, J., WILLIAMS, L.: *Hidden in Plain Sight: Automatically Identifying Security Requirements from Natural Language Artifacts*,

The 2-nd IEEE International Requirements Engineering Conference: Proceedings, 2014, pp. 183-192.

[135] IWAMA, F., NAKAMURA, T., TAKEUCHI, H.: *Constructing Parser for Industrial Software Specifications Containing Formal and Natural Language Description*, The 34-th International Conference on Software Engineering: Proceedings, 2012, pp. 1012-1021.

[136] GNESI, S., LAMI, G., TRENTANNI, G., FABBRINI, F., FUSANI, M.: *An automatic tool for the analysis of natural language requirements*, Computer Systems Science and Engineering, 2005, vol. 20 (1), pp. 53-62.

[137] SIEGEMUND, K.: *Contributions to Ontology-Driven Requirements Engineering: Dissertation*, Dresden: Technischen Universität Dresden, 2014.

[138] FARFELEDER, S., MOSER, T., KRALL, A., STALHANE, T., OMORONIYA, I., ZOJER, H.: *Ontology-Driven Guidance for Requirements Elicitation*, Lecture Notes in Computer Science, 2011, vol. 6644, pp. 212-226.

[139] MURTAZINA, M., AVDEENKO, T.: *The ontology-driven approach to support the requirements engineering process in Scrum framework*, CEUR-WS, 2018, vol. 2212, pp. 287-295.

[140] MUSTAFA, A., WAN-KADIR, W. M. N., IBRAHIM, N., SHAH, A., YOUNAS, M.: *Integration of Heterogeneous Requirements using Ontologies*, International Journal of Advanced Computer Science and Applications, 2018, vol. 9 (5), pp. 213-218.

[141] SU, J., SACHENKO, A., LYTVYN, V., VYSOTSKA, V., DOSYN, D.: *Model of Touristic Information Resources Integration According to User Needs*, The 2018 IEEE 13-th International Scientific and Technical Conference on Computer Sciences and Information Technologies: Proceedings, 2018, vol. 2, pp. 113-116.

[142] HOVORUSHCHENKO, T., POMOROVA, O.: *Ontological Approach to the Assessment of Information Sufficiency for Software Quality Determination*, CEUR-WS, 2016, vol. 1614, pp. 332–348.

[143] HOVORUSHCHENKO, T.: *Models and Methods of Evaluation of Information Sufficiency for Determining the Software Complexity and Quality Based on the Metric Analysis Results*, Central European Researchers Journal, 2016, vol. 2, issue 2, pp.42-53.

[144] HOVORUSHCHENKO, T.: *Information Technology for Assurance of Veracity of Quality Information in the Software Requirements Specification*, Advances in Intelligent Systems and Computing, 2018, vol. 689, pp. 166–185.

[145] ISO/IEC/IEEE 29148:2018. Systems and software engineering. Life cycle processes. Requirements engineering. [Introduced 01.12.2018]. Geneva (Switzerland), 2018. 28 p. (International standard).

[146] HOVORUSHCHENKO, T.: *Forming the Logical Conlcusion about Sufficiency of Information of Software Requirements Specification for Software Quality Assessment*

by ISO 25010:2011, IEEE First Ukraine Conference on Electrical and Computer Engineering: Proceedings (Kyiv, May 29 – June 2, 2017), pp. 789-794.

[147] HOVORUSHCHENKO, T.: *The Rules and Method of Forming the Logical Conclusion about Sufficiency of Information for Software Metric Analysis*, 2017 IEEE International Scientific and Technical Conference on Computer Science and Information Technologies: Proceedings (Lviv, September 5-8, 2017), pp.7-11.

[148] HOVORUSHCHENKO, T., PAVLOVA, O., BOYARCHUK, A.: *Modelling of non-functional characteristics of the software for selection of accurate scope of information for their evaluation*, CEUR-WS, 2019, vol. 2533, pp. 206-216.

[149] HOVORUSHCHENKO, T., PAVLOVA, O.: *Evaluating the Software Requirements Specifications Using Ontology-Based Intelligent Agent*, 2018 IEEE International Conference on Computer Science and Information Technologies: Proceedings (Lviv, Ukraine, September 11-14, 2018), vol.1, pp. 215-218.

[150] HOVORUSHCHENKO, T., PAVLOVA, O., FEDULA, M.: *Improving the input information for medical software requirements specifications using ontology-based intelligent agent*, CEUR-WS, 2018, vol. 2255, pp.113-125.

[151] HOVORUSHCHENKO, T., PAVLOVA, O.: *Method of Activity of Ontology-Based Intelligent Agent for Evaluating the Initial Stages of the Software Life Cycle*, Advances in Intelligent Systems and Computing, 2019, vol. 836, pp. 169-178.

[152] 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, 2019, vol. 1, no. 2 (97), pp. 6-17.

[153] HOVORUSHCHENKO, T., PAVLOVA, O.: *Intelligent System for Determining the Sufficiency of Metric Information in the Software Requirements Specifications*, CEUR-WS, 2019, vol. 2353, pp.253-266.

[154] HOVORUSHCHENKO, T., PAVLOVA, O., MEDZATYI, D.: *Ontology-Based Intelligent Agent for Determination of Sufficiency of Metric Information in the Software Requirements*, Advances in Intelligent Systems and Computing, 2020, vol.1020, pp. 447-460.

[155] HOVORUSHCHENKO, T., LOPATTO, I., PAVLOVA, O.: *Concept of Intelligent Agent for Verification of Considering the Subject Area Information*, 2020 IEEE 11th International Conference on Dependable Systems, Services and Technologies: Proceedings (Kyiv, Ukraine, May 14-16, 2020), pp. 465-469.

[156] BOYARCHUK, A., PAVLOVA, O., BODNAR, M., LOPATTO, I.: *Approach to the Analysis of Software Requirements Specification on Its Structure Correctness*, CEUR-WS, 2020, vol. 2623, pp. 85-95.

prof. Ing. Tetiana Hovorushchenko, DrSc., Ing. Olga Pavlova,
doc. Ing. Artem Boyarchuk, PhD., doc. Ing. Miroslav Kvassay, PhD.,
doc. Ing. Yelyzaveta Hnatchuk, PhD., doc. Ing. Dmytro Medzatyi, PhD.

Intelligent Information-Analytical Technologies for Improving the Software Quality by Assessing the Sufficiency of Information at Initial Stages of the Life Cycle

Printed by EDIS-Žilina University Publisher, 2020
First edition, 100 copies, AA 13,80
ISBN 978-80-554-1729-5