# The ontology-driven approach to support the requirements engineering process in Scrum framework

**M Sh Murtazina[1] and T V Avdeenko[1]**

[1]Novosibirsk State Technical University, Karla Marks ave 20, Novosibirsk, Russia, 630073

**Abstract**. The paper presents an approach to the Support of the requirements engineering process in the field of software development process by applying OWL ontology. A brief overview of the capabilities of using ontologies for intellectual support of the process of requirements engineering is given. The main features of requirements engineering for Scrum project management of software development are analyzed.The quality assessment criteria of user stories are explored. The developed ontology accumulates knowledge about the quality assessment criteria of user stories, about requirements artifacts, types of requirements, about elements of Scrum framework. The ontology includes axioms that determine the quality of user story expression and the quality properties of the requirements.Also ontology includes axioms determining the priority and risk of user stories.The ontology is implemented in the Protégé environment.



## 1. Introduction

The International Standard ISO/IEC/IEEE 29148-2011 *Systems and software engineering – life cycle processes – requirements engineering*determines requirements engineering as "interdisciplinary function that mediates between the domains of the acquirer and supplier to establish and maintain the requirements to be met by the system, software or service of interest" [1, p.8]. A complex system of requirements is developed and maintained in the process of requirements engineering. This system of requirements is built on the basis of needs, expectations, constraints and interactions of the stakeholders. Under the stakeholders the Standard ISO/IEC/IEEE 29148-2011 primarily implies users and customers.The circle of stakeholders also includes software developers and suppliers.

Requirements engineering plays a key role in ensuring the success of the software development. Eliciting and managing requirements is an extremely difficult task for any methodology of software development project management. Project management methodologies are usually subdivided into rigid and agile approaches. A rigid approach to the management of the software development project is characterized by detailed planning. Requirements engineering here is a separate stage that precedes all other stages of the software building. In case of agile methodologies planning is performed only for the current iteration. Requirements engineering here is an iterative process where the requirements constantly evolve. Changeability of requirements is a serious problem for software development projects. In this regard, rigid methodologies are much inferior to agile ones. With an agile approach, precious time is not wasted on trying to anticipate all possible requirements and document them in detail. A typical form of agile high-level requirements in agile methodologies is feature requests and user stories. They are formulated as one or more sentences which illustrate the user's goals that a

software function will satisfy. The details are clarified when the requirements are implemented within the regular iteration. This happens in the process of active interaction with the stakeholders. Incompleteness that is inherent to requirements specification in agile methodologies is compensated by extensive informal communication with stakeholders. Another problem area is to ensure the consistency of the requirements coming from different stakeholders and also to reveal the same requirements presented by different stakeholders with different terminology.

Scrum is one of the most popular and well-developed agile methodologies. According to the Internet survey of Agile Survey, only in 2017 this methodology was used by 56% of the respondents' organizations[2]. Despite the immense popularity of agile methodologies in general, and the Scrum methodology in particular, development of approaches to assessing the quality of specifications for agile requirements is still relevant. Many existing investigations use the INVEST model proposed in 2003 by B. Wake [3]. New approaches to assessing the quality of requirement specifications for agile methodologies began to appear in the mid-2010s. Examples of new approaches are Quality User Story Framework [4] and Agile Requirements Quality Framework [5].

The trend of recent years in the field of supporting the process of requirements engineering and assessment of the software requirements specification quality is knowledge-based systems built on the ontologies. The ontological approach is promising for overcoming some of the deficiencies in requirements engineering [6]. However the existing ontology-base solutions are more focused on assessing the requirements specifications in accordance with international standards and do not take into account the specifics of Scrum and the evaluation of the quality of user stories.

In the present paper we propose an ontology-driven approach to support the requirements engineering process in Scrum framework. The paper is organized as follows. In Section II the literature review is made. Section III analyzes the features of the requirements engine in Scrum. Section IVdiscusses the issues of assessing the quality of user stories.Section V proposes an ontological model for supporting the process of requirements engineering in Scrum. In section VI conclusionsabout prospects for the proposed approach are made.

## 2. Background

In the early 2000s K. K. Breitman et al. considered the ontology development process as a subprocess of the process of requirements engineering [7]. In [8] it was proposed to use the domain ontology as an infrastructure for refinement of software requirements. G. Dobson et al. pointed out that ontologies are useful for the representation and interconnection of many types of knowledge. Requirements engineering involves the extraction of knowledge from a variety of sources. Ontologies in requirements engineering can be used for representation of the requirements model itself, as well as acquisition structures for domain knowledge, the application domain and the environment [9].

One of the first works, in full demonstrating the possibility of the ontologies for the Requirements engineering, is the work of M. Kossmann et al. in which semi-automated methodology OntoREM (Ontology-driven Requirements Engineering Methodology) was developed [10]. Methodology OntoREM includes processes, methods and tools. The aim of this methodology is to create requirements specifications for systems in less time and at lower costs while improving the quality of such specifications. Methodology OntoREM was applied by its authors in the company Airbus to develop aircraft operability requirements. Using OntoREM resulted in significant savings of money and time [11]. The structure of the OntoREM process developed by M. Kossmann et al. is shown in Figure 1.

To provide the OntoREM process, Kossmann M. et al. used the MindManager Tool Environment, the OntoRAT Tool Environment, the Protégé Tool Environment, and the DOORS Tool Environment. First, the domain ontology intheformofaMindmap is developed using in MindManager. This tool is convenient for quick visualization of the domain ontology and allows saving the result in the OWL format. The Protégé environment is used to manage the ontologies obtained with MindManager. The OntoRAT tool (Ontology-driven Requirements Analysis Tool) is used to analyze requirements by status, purpose, soft targets (i.e. targets without clear criteria) and traceability. The OntoRAT tool was developed during the OntoREM project. The IBM Rational DOORS software package is used for requirements management. The OntoREM project has, to a large extent, led to an active analysis of the

possibilities of using of ontologies in many subject domains including the field of software development.

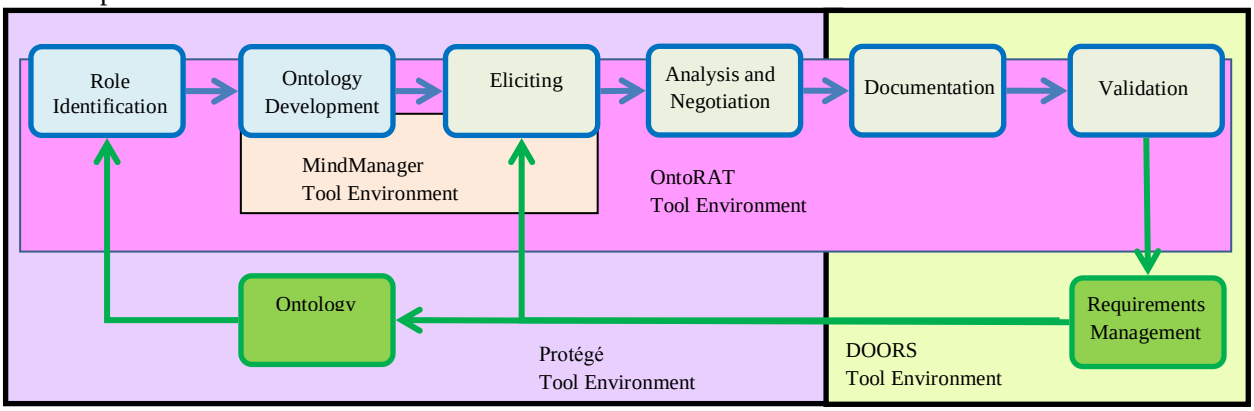


**Figure 1.** OntoREMprocess [10].

In paper [6] an approach to automating the process for quality evaluation of requirements is presented. B K. Siegemund has distinguished two types of support for the requirements engineer: "(1) support for the specification of the requirements knowledge and (2) validation and error elimination support" [6, p.79]. Support for requirement engineering is provided by two ontologies: "Guidance Ontology" and "Requirements Ontology". The GORE (Goal-Oriented Requirements Engineering) method is used in the approach developed by K. Siegemund. A set of optional and mandatory tasks based on validation rules and their pre- and postconditions are contained in the Guidance Ontology. The basis of the Requirements Ontology is the IEEE 830-1998 standard. The Requirements Ontology accumulates knowledge about the requirements for a particular project

The approach based on frame ontology is proposed in [12-14]. Application of the ontology allows building a harmonized model of requirements for software development process. This is designed to help the analyst to take into account all aspects of the requirements. The theory of the field structure of speech parts and recommendations of the SWEBOK were applied to construct the model. Templates of the specification structure were designed in the Protégé environment.

A metamodel is proposed in [15] which is used to reason about the requirements. This model is implemented as an OWL ontology. The model includes requirements, requirement artifacts and stakeholders. The requirements are linked by four types of relations: Refines, Requires, Conflicts, and Contains. These relations allow building rules to reason about requirements traceability, consistency and completeness.

An ontological approach to improving the requirements engineering in the agile development is presented in [16]. C. Thamrongchote et al. pointed out that although the templates of user stories are easy to use, the application of the domain terminology when writing them is a difficult problem. It is proposed in [16] to accumulate user stories from previous successful projects in the ontology knowledge base, in order to improve the process of subsequent working with them. The ontology schema is designed using class and hierarchy relations. Hierarchy relations are used, for example, to determine which user roles inherit features from other roles. For example, the class "Guest" is a subclass of the "Customer" class. Accordingly, the set of user stories of the "Guest" should be implemented for the "Customer". Also ontology establishes synonymous relations. For example, a synonymous relation exists between the "Guest" and "Visitor" classes. This means that user stories that mention the roles "Guest" and "Visitor" describe the features for users of the same class. The words extracted from the user story are distributed in three classes: "Role", "Action" and "Object". Individuals of the classes "Role" and "Action" are associated with the relation "perform Action" and individuals of the classes "Action" and "Object" – relation "perform Object".

The authors of paper [17] propose to use the ontology to predict the effort estimates in the implementation of user stories. The accuracy of effort estimates depends on the level of analysis of the software development context and the experience of persons making the evaluation. The approach

developed by M. Adnan et al. Proposes to accumulate unique knowledge of key project participants. This knowledge is then used to evaluate efforts to implement user stories.

The analysis of scientific publications shows that the questions of designing the ontology-driven approach to support the requirements engineering process in the Scrum framework are not yet sufficiently developed. The considered complex ontology-oriented approaches do not take into account the peculiarities of working with user stories. At the same time, research in the field of requirements engineering in agile projects focuses on certain aspects of working with user stories and evaluation of their quality.

## 3. Requirements engineering in Scrum framework

According to the Guide [18], Scrum is a process framework designed for developing and sustaining complex products. The core of the Scrum is *Sprint*, that is a time interval of one month or less, during which the Scrum team creates a potentially releasable product *Increment*. The basic elements of the Scrum framework are depicted in Figure 2. These include the Scrum Team, the Scrum Events and the Scrum Artifacts.

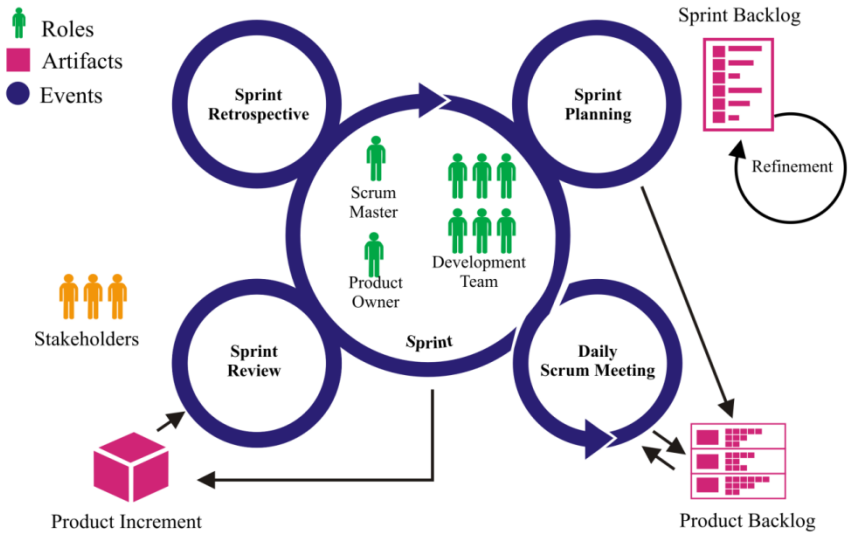


**Figure 2.** The basic elements of the Scrum framework.

Requirements engineering in Scrum is an iterative process. The requirements evolve in each Sprint [19, p.27]. Eliciting the requirements-needs occurs during the Sprint Review. The next stage is a requirements analysis accepted for execution in the Sprint. From the point of view of the Scrum Team the requirements should be unambiguous, complete and consistent. Conflicts can be resolved by prioritizing user stories or overriding incorrectly formulated requirements.

Documenting the requirements helps to analyze and verify the requirements specification. Although documentation in agile software development methodologies is used in smaller volumes than in rigid methodologies, requirements documentation is an essential part of the work process. Allocate the following base forms of requirements records: feature requests and stories. The following forms of recording the requirements in agile methodologies are allocated: feature request and story. Feature request is a structured query (with header, description and set of attributes) for new or refined feature of the software. Story is a high-level requirement formulated as one or more sentences in the user's everyday or business language so that the developers can give a reasonable estimate of an effort to implement it. The following types of stories are distinguished by the form of the record:

- userstory;
- technical story / technical user story;
- jobstory.

The most common form to record the requirements when applying the Scrum framework is user story. According ISO / IEC / IEEE 26515-2011 user story is “simple narrative illustrating the user goals

that a software function will satisfy" [20, p.3]. In accordance with this standard user story has to include [20, p.18]:

- role of the user;
- goal that the user achieves;
- value for the customer;
- acceptance criteria that allow to determine that the user story is implemented.

The following recording scheme is usually used in Scrum project for the user story:

*As a <type of user X >*
*I want <some goal Y>*
*So that <some reason Z>*

The last part of the user story should show the user's benefit from using the feature (the reason why this story is needed by the user). In Scrum, a detailed analysis of user story occurs at the moment of implementing the feature to which it relates. During the Sprint planning, the user story usually is used as the basis for a conversation between the Development Team and the Product Owner in order to clarify the details of the implementation. The acceptance criteria for the user story are determined by the Development Team during the discussion.

Acceptance criteria are a set of statements that specify both functional and non-functional requirements. It should be noted that attention is usually focused on functional requirements so many non-functional requirements may not be taken into account. Acceptance criteria can be written in a variety of formats. There are two main approaches to the recording of acceptance criteria: rule-oriented or scenario-oriented ones.Gherkin notation can be used when using a scenario-oriented approach.

During the Daily Scrum meetings, it is discussed what was done during the last working day and what difficulties arose. Each member of the Development Team notifies the Team what they are going to implement for the current day. In a Daily Scrum, the problems found in the requirements for the software product can be discussed and a strategy for their solution can be worked out.

Sprint Review is an event that takes place at the end of Sprint to inspect the Increment and adapt the Product Backlog if needed. Validation of the requirements occurs during the Sprint Review. Requirements management is based on the stakeholder feedback during the Sprint Review. The Sprint Retrospective is an event when the Scrum Team analyzes the work done and creates a plan for improvements to be enacted during the next Sprint. These improvements relate to all work processes including engineering requirements.

## 4. Quality assessment of user stories

There are many approaches to quality assessment of requirements. The standard ISO/IEC/IEEE 29148:2011 defines quality criteria for the individual requirements and sets of requirements. Each individual requirement must be necessary, implementation free, unambiguous, consistent, complete, singular, feasible, traceable, verifiable. The set of requirements must possess the following quality characteristics: completeness, consistency, affordability, boundedness [1, p.11-12].

The QUS (Quality User Story) Framework proposed by Lucassen et al. in [4] can be used to assess the quality of user stories. The structure of the criteria in the QUS framework is based on the understanding of quality in categories O.I. Lindland. There are syntactic quality, semantic quality and pragmatic quality. Syntactic quality criteria are used to evaluate the textual structure of a user story without considering its meaning. These criteria are used to quality assessment of individual user stories. In the structure of user story, three parts are identified: a role, a means and an ends (optionally). The group of syntactic quality criteria includes:

- well-formed: in the user story test there is at least a role and a means;
- atomic: the test user story expresses a requirement for exactly one feature of the software;
- minimal: in the user story test there is nothing except a role, a means and the ends.

Semantic quality criteria are used to evaluate "the relations and meaning of (parts of) the user story text" [4]. The group of semantic quality criteria includes:

- conflict-free: a user story should not conflict with any other user story;

- conceptually sound: the means expresses a feature and the ends expresses a rationale;
- problem-oriented:a user story determines the problem, not the solution to it;
- unambiguous: the text of user story does not contain terms or abstractions that can lead to multiple interpretation.

The criterion "conflict-free" is used to quality assessment of a set of user stories. The other three criteria are used to quality assessment of individual user stories.

Pragmatic quality criteria affect the “choosing the most effective alternatives for communicating a given set of requirements”[4]. The group of pragmatic quality criteria includes:

- fullsentence: a user story is a well-formed full sentence;
- scalable: a user story does not specify the coarse-grained requirements that are difficult to plan and prioritize;
- unique: each user story is unique, duplicates are not allowed;
- uniform: all user stories in the specification use the same template;
- independent: a user story is self-contained and does not have inherent dependencies on other user stories;
- explicit dependencies: if a user story has non-obvious dependencies on others, explicit references should be made to the stories from which there is a dependency
- complete: implementing a set of user stories creates a feature-complete application, no steps are missing (for critical user stories).

The "full sentence" and "scalable" criteria are used to quality assessment of individual user stories. The remaining criteria are used to quality assessment of a set of user stories.

## 5. Ontology for requirementsengineering process in Scrum framework

OWL ontology to support the engineering requirements process is implemented in the Protégé environment. This ontology accumulates knowledge about the key features of the requirements engineering in the Scrum framework. The taxonomy of the upper level classes, which are direct descendants of the general *Thing class*, is shown in Figure 3.

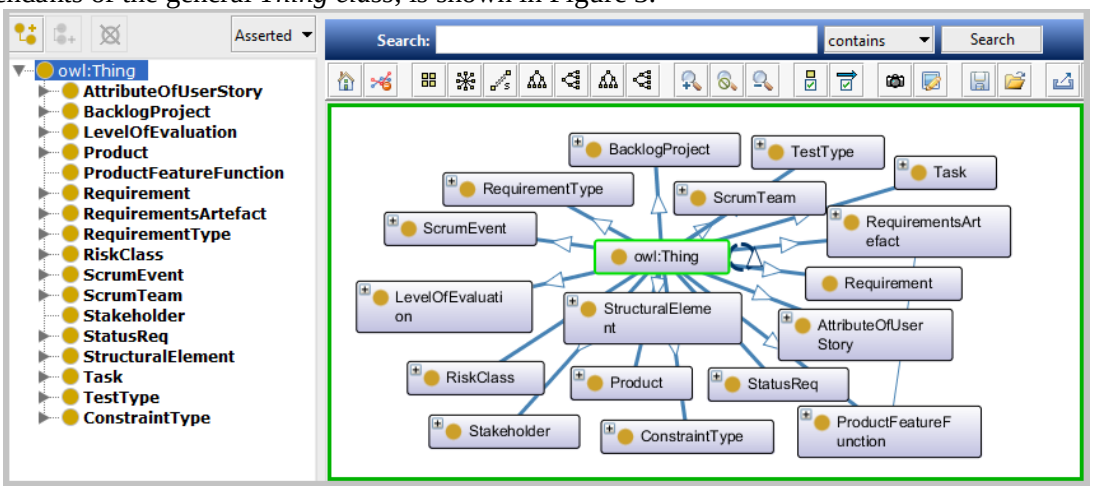


**Figure 3.** The taxonomy of the upper level classes.

The class *Attribute Of User Story* describes the attributes of user stories such as priority, risk level, etc. The class *Backlog Project* is a class whose subclasses are the types of backlogssuch as Product Backlog, Sprint Backlog and Task Backlog. The class *Level Of Evaluation*is a class whose subclasses are qualitative scales used in assessing priorities, risks, etc. The concept *Product* corresponds to the term "software product". The concept *Product Feature Function* corresponds to the term "function of the software". The class *Requirement* contains subclasses corresponding to the term "requirement". The development of ontology is based on the following understanding of the term "requirement". A requirement is an assertion about some property of the software product. For example, user story, acceptance criteria fixed with use of rule-oriented or scenario-oriented approaches. The class

*Requirements Artefact*contains subclasses corresponding to the concept "requirement artefact". Since such requirements as *Scenarios*, *Acceptance criteria* and *Definition of done* represent information also about the requirements that are created, modified and used in the process of implementing the requirements, they are simultaneously included in the classes *Requirement* and *Requirements Artefact*. Also subclasses of the class *Requirements Artefact* are requirements sources and software features and tests that verify the correctness of the requirements implementation.

The class *Requirement Type* contains requirements classification (for example, functional and non-functional requirements). The class *Risk Class* contains subclasses describing the classification of risks such as a risk of getting a bug, security risk and etc. The class *Scrum Event* contains subclasses corresponding to the Scrum events from the Scrum Guides. The class *Scrum Team* contains subclasses corresponding to the Scrum roles from the Scrum Guides. The class *Stakeholder* corresponds to the term "*stakeholder*". The class *Status Req* contains subclasses describing the status of requirements such as “resolved”, “in progress” and etc. The class *Task* is a class whose instances are tasks performed by the development team. The class *Test Type* contains subclasses describing the classification of test types (for example, performance testing). The class *Constraint Type* contains subclasses describing the types of constraints for non-functional requirements. Constraints express requirements to the internal and external quality of the software product. The class *Structural Element* contains subclasses describing structural elements that must include requirements written using some technique. Figure 4 shows the relations between the class *UserStory* and the classes that describe its structure.

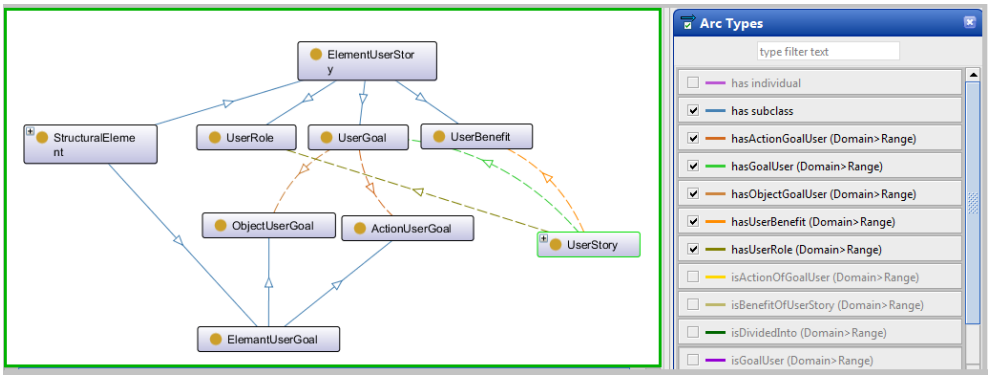


**Figure 4.** A conceptual structure of user story.

The structure of the user story includes the user role, user goal and the user benefits (or in other words, the reason of the occurrence of the story). The user goal in turn consists of the action that the user makes and the object on which the action is performed. For example, "As a user, I want to sort photos, so that easy to view photos". In this example, the user goal consists of the action "sort" and the object "photos". Theuserbenefitis "easytoviewphotos".

In the ontological model, rules for assessing the quality of individual user stories and sets of user stories are introduced. For example, consider the criterion for evaluating syntactic quality – “Well-formed user story”. A user story is considered well-formed if it specifies the role, goal and benefits. The goal should include the action and the object:

IF User Story has (User Role and Well Formed Goal User and User Benefit) THEN User Story is Well Formed User Story.

IF Goal has (Action Goal User and Object Goal User) THEN Goal is Well Formed Goal User.

The implementation of these rules in the axioms of the ontology in the Protégé environment is shown in Figure 5.

To extract instances of the classes User Role, Action User Goal, Object User Goal and User Benefit from the user story text, morphological, syntactic and semantic analysis of the text can be performed using the appropriate utilities.

A separate user story satisfies the completeness property (internal completeness), if it is included all the information necessary to ensure correct implementation. This means that the user story must be evaluated and acceptance criteria for it must be established, for user story should be indicated the

source and the team member who added it in the baglog. The assessment of user story includes effort estimates, priority and assessing the level of risk. The proposed ontology axiom for assessing the completeness of user story is shown in Figure 6.

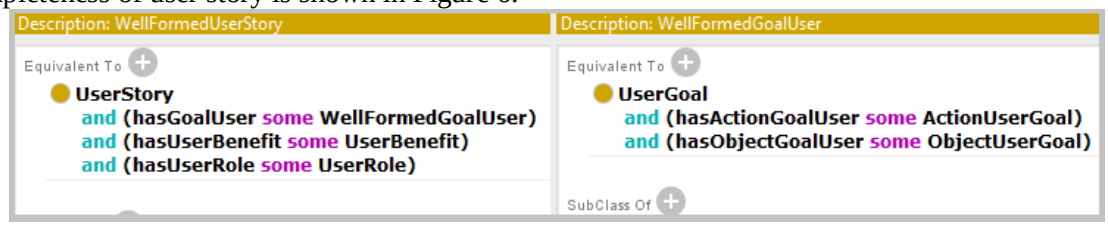


**Figure 5.** Axioms for evaluating the structure of user story.

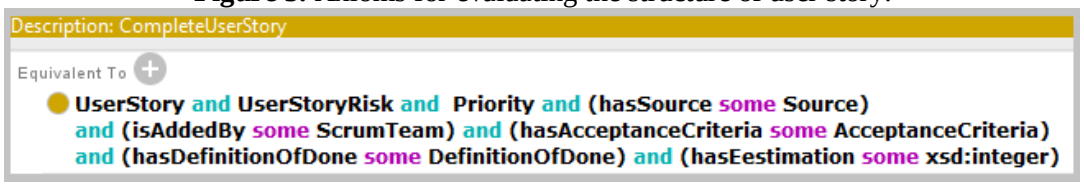


**Figure 6.** Axioms for evaluating the completeness of user story.

If information is entered to calculate the level of risk then a user story is included in the class *User Story Risk*. If information is entered to calculate the priority of the user story then the user story is included in the class *Priority*. Consider the priority evaluation. In the proposed approach, the user story priority is set depending on the entered parameters – business value and urgency. The simplest from the point of view of the organization of the assessment process is ranking by qualitative scale. In the developed approach, the scale contains four divisions: "critical level" (4), "high" (3), "middle" (2) and "low" (1). The matrix for priority estimation and an example of an axiom for determining a low priority level are shown in Figure 7.

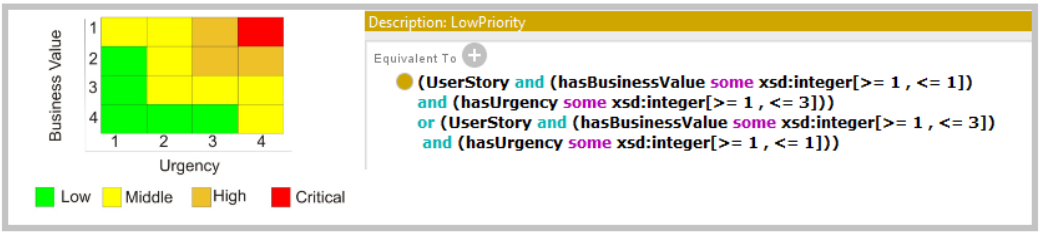


**Figure 7.** Matrix for priority estimation and an axiom for determining a low priority level.

The developed ontology allows checking the quality criteria of individual user stories. In practice, the definition of completeness and conflictuality of a set of user stories is a context-dependent problem that is difficult to generalize. However, you can talk about the incompleteness of the set of user stories if the user stories meet the requirements for manipulating the elements of the system that were not previously created. You can talk about the conflict of a set of user stories if for one object from one type of user there is permission and lockout the same action. The latter can be partly solved by constructing the domain ontology from the structural elements of user stories and behavior scenarios detailing user stories. Methods of extracting knowledge units from a set of texts can be used for semi-automatic construction of domain ontology (for example, as in [21]).

## 6. Conclusion

The paper shows that the application of an ontology-oriented approach to supporting the process of requirements engineering in Scrum is an extremely urgent task. Requirements are the raw data for software development and must satisfy certain quality characteristics. Currently, the main characteristics of the requirements quality for software products are defined by the standard ISO/IEC/IEEE 29148-2011 *Systems and software engineering – life cycle processes – requirements engineering*. The analysis of scientific publications allows us to talk about the need to apply assessment models that take into account the specifics of agile requirements.

In present paper the ontology was presented taking into account the features of the Scrum framework and the quality criteria that are characteristic for the user stories. In analyzing the features of the requirements engineering in the Scrum framework found that the main effort is focused on the analysis of the functional component, so non-functional requirements are often not documented. Taking into account high frequency of updating the requirements when working in accordance with the Scrum framework, application of the proposed approach allows to quickly monitoring traceability and completeness of the requirements. Making records in the ontological knowledge base will also serve as a good tool for documenting the progress of the work. The latter can help to increase the productivity of the development team.

## 7. References

[1] *ISO/IEC/IEEE 29148:2011(E), Systems and software engineering – life cycle processes – requirements engineering* (Version 1.0, 2011-12-01).
[2] VersionOneInc 2018 *The 12th annual State of Agile report* (Access mode: https://explore.versionone.com/state-of-agile/versionone-12th-annual-state-of-agile-report)
[3] Wake B 2003 *INVEST in Good Stories, and SMART Tasks* (Access mode*:* https://xp123.com/articles/invest-in-good-stories-and-smart-tasks/)
[4] Lucassen G, Dalpiaz F, van der Werf J M and Brinkkemper S 2015 *Proc. Int. Conf. on Requirements Engineering* 126-135
[5] Heck P and Zaidman A 2014 *Preprint arXiv*: *1406.4692*
[6] Siegemund K 2014 *Contributions To Ontology-Driven Requirements Engineering : dissertation to obtain the academic degree Doctoral engineer* (Dresden: Technischen Universität Dresden)
[7] Breitman K K and Leite J C S P 2003 *Proc. Int. Conf. on Requirements Engineering* 309-319
[8] Zhu X and Jin Z 2005 *Proc. Int. Conf. onEngineering of Complex Computer Systems* 402-410
[9] Dobson G and Sawyer P 2006 *Int. Seminar on Dependable Requirements Engineering of Computerised Systems at NPPs* (Halden: Institute for Energy Technology)
[10] Kossmann M, Wong R, Odeh M and GilliesA 2008 *Proc. Int. Conf. on Information and Communication Technologies: From Theory to Applications*
[11] Kossmann M, Odeh M, Gillies A and WattsS 2009 *Proc. Int. Conf. onApplications of Digital Information and Web Techn* 95-103
[12] Pustovalova N V and Avdeenko T V 2016 *SPIIRAS Proceedings* **1(44)** 31-49
[13] Avdeenko T V and Pustovalova N V 2016 *Proc. Int. Conf. onActual problems of electronic instrument engineering* **1** 513-518
[14] Avdeenko T V and Pustovalova N V 2015 *Proc. Int. Siberian conference on control and communications*
[15] Goknil A, Kurtev I, van den Berg K 2008 *Proc. Int. Conf. onModel Driven Architecture – Foundations and Applications* 310-325
[16] Thamrongchote C and Vatanawood W 2016 *Proc. Int. Conf. onComputer and Information Science* 633-636
[17] Adnan M and Afzal M 2017 *IEEE Access* **5** 25993-26005
[18] Schwaber K and Sutherland J 2017 *The Scrum Guide* (Access mode: https://www.scrumguides.org /docs/scrumguide/v2017/2017-Scrum-Guide-US.pdf#zoom=100)
[19] Darwish N D and Megahed S 2016 *International Journal of Computer Applications* **149(8)** 24-29
[20] ISO/IEC/IEEE 26515:2011(E), Systems and software engineering *Developing user documentation in an Agile environment (Version 1.0, 2011-12-01)*
[21] Mikhaylov D V, Kozlov A P and Emelyanov G M 2016 *Computer Optics* **40(4)** 572-582 DOI: 10.18287/2412-6179-2016-40-4-572-582

**Acknowledgments**
The reported study was funded by Russian Ministry of Education and Science, according to the research project No. 2.2327.2017/4.6.