An Ontology-Based Approach to the Agile Requirements Engineering

Marina Murtazina(✉) and Tatiana Avdeenko

Novosibirsk State Technical University, 630073 Novosibirsk, Russia
murtazina@corp.nstu.ru

Abstract. The paper presents an approach to the agile requirements engineering based on the OWL ontologies. A brief overview of the benefits of an ontology-based approach to requirements engineering is given. Attention is focused on agile engineering requirements process. The proposed approach uses three ontologies. The first ontology is used to represent knowledge about the agile requirements engineering process. The second ontology is designed to match natural language sentences with the requirements in order to identify conflicts. The third ontology is used to accumulate the knowledge about the domain of the software product. The first ontology is core. This ontology consists of classes corresponding to events, roles and artefacts of agile development. Object properties established between the individuals of class can be used to identify directly or indirectly linked requirements and requirements artefacts. This enables maintaining requirements traceability. Also the ontology takes into account particular qualities of working with the requirements in agile development processes including knowledge about the criteria for assessing the quality of user stories that is the most common form to record the requirements in agile methods. The ontologies are implemented in the Protégé environment.

Keywords: Requirements engineering · Ontology · Agile environment

1 Introduction

The success of software products depends on the extent to which they, as tools, can be effectively used in the implementation of the end user tasks. That is why requirements engineering plays a key role in the software development. The requirements engineering as a field of knowledge includes elicitation, analysis, specification and validation of the requirements [1]. By their nature, the software requirements represent complex knowledge that is extracted in the process of requirements engineering from various sources, including many stakeholders, whose views on the developed product can be diametrically opposed. In this regard, the requirements can be considered as a result of alternative solutions in the field of determining functional and qualitative characteristics of the software.

The decision making process in requirements engineering depends greatly on the experience and intuition of the development team. In particular, the team members use their experience in analyzing feasibility and determining complexity of the requirements, identifying inconsistencies and incompleteness in the requirements sets, forecasting implementation deadlines, assessing implementation risks, etc. Frequent modifications of the requirements inherent to the agile development methodologies as a reaction to changes in the business environment, requires permanent monitoring on the consistency of the requirements specification and the actual priority of the software product functions. Therefore, the ability to quickly identify and resolve conflicting requirements, and also revise priorities to meet new requirements, is critical to the development of the software development project.

The scientific discourse of recent years is characterized by a focus on the application of ontological models to the software development process. Ontologies provide a formal representation of knowledge and relationships between the concepts used in the software development. Ontologies can be used for requirements analysis and specification phases [2]. Ontologies allow to expand the possibilities of model-driven requirements engineering (MDRE) through the use of machine reasoning. Over the past few years ontology-driven requirements engineering (ODRE) has become a leading trend [3].

The success of the ontological approach in the requirements engineering is determined by its capabilities, such as availability of the domain vocabulary, formulating knowledge about the application area and its reuse, understanding the problem area, improving communication between specialists from different fields [4]. Ontologies in the requirements engineering are used to formalize the structure of documents for working with the requirements, to represent the types of the requirements and knowledge about the domain of the software product [5]. Ontologies also allow formulating the rules (axioms) for reasoning about traceability, consistency and completeness of the requirements [6]. The ontological approach is useful for comparing stakeholders' points of view on different subsystems of a single information system [7].

Ontologies can be used to improve the requirements development [8] and requirements management [9] in agile software development process. Ontological approach to the requirements engineering allows to improve the process of user story formulation [10], to facilitate verification of compliance with the quality characteristics of individual user stories and sets of user stories [11], as well as to obtain more accurate assessments of the efforts required to implement user stories [12]. Summarizing the above, it should be noted that most studies of the possibilities of using ontologies in requirements engineering do not take into account the specifics of the project management life cycle or the software development life cycle. To the best of our knowledge, very few papers apply ontology-based approach to the agile project development. However, it seems that the application of the ontology-base approach can be especially valuable for agile development. It is precisely under conditions of permanent changes in the requirements and their priorities inherent to the agile development that knowledge engineering methods prove to be especially useful. Representation of knowledge about the project requirements in the form of ontologies allows the use of machine reasoning methods that can be used for discovering logical inconsistencies.

In the present paper we propose an ontology-based approach to the agile requirements engineering using a system of three ontologies. The first ontology is needed to represent knowledge about the agile requirements engineering process, the second one is designed to match natural language sentences with the requirements in order to identify conflicts, the third ontology is used to accumulate the knowledge about the domain of the software product. The paper is organized as follows. In Sect. 2 we provide a review of the research topic and the terminology used. In Sect. 3 we define the ontology for agile requirements engineering process. In Sect. 4, we give conclusions about the prospects of using this approach in agile software development.

2 Theoretical Background

2.1 Requirements Engineering in Agile Software Development

In agile development, the software requirements specification (SRS) is an integrated requirements package which is maintained up to date. This package is not a single physical document, but a logical structure filled with requirements for a software product.

To date, the Scrum framework is the most popular among the agile project management framework. The basis of Scrum is the theory of empirical control. According to this theory, the source of knowledge is experience, and the source of solutions is real data. The basic scheme of the Scrum framework works is presented in Fig. 1.

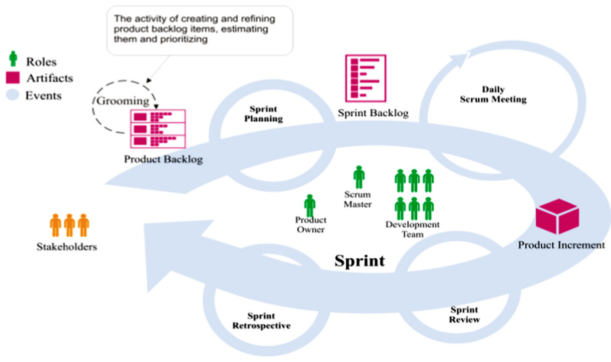


Fig. 1. The basic scheme of the Scrum framework works

According to Agile Manifesto [13], agile development encourages the creation of the minimum amount of documentation necessary to manage and coordinate the project participants work in developing a software product. When a software project is launched, it is not necessary to create a comprehensive requirements document. First, the document "Vision and Scope" is developed. This document determines the stakeholders vision on the software being developed in terms of the key needs and constraints under which the project will be implemented.

Next, work begins on filling the Product Backlog, which is a prioritized list of currently existing product requirements, which is never complete. Product backlog items can be divided into product features, defect, technical work, knowledge acquisition by type of work [14]. Product feature is a unit of functionality of a software product that satisfies a requirement. The remaining three types of product backlog items are needed to plan work on eliminating defects, refactoring, database migration, research necessary to implement the requirements of any type, etc.

Before a feature is scheduled for a sprint, it is necessary to decompose it into small user stories, develop basic behavior scenarios, evaluate implementation efforts, identify dependences on other user stories, and determine the priority. It is also necessary to analyze low priority user stories. Perhaps these product backlog items became important, or, on the contrary, so unimportant that they should be removed. This work is done by the Product Owner together with the development team as part of the Backlog grooming.

Backlog grooming (Product backlog refinement or grooming) is an activity throughout the sprint aimed at revising the backlog of the product to prepare its product for the next sprint planning. Backlog grooming helps ensure that the requirements will be clarified, and user stories will be prepared for work in advance of planning for the sprint. As a result of Backlog grooming, the top of the Product Backlog should include user stories prepared for sprint. In this case, such user stories should be enough for 2–3 sprints. User stories should be clear to all team members, evaluated by the team, and acceptance criteria should be indicated for the stories. The acceptance criteria can be written in the form of simple sentences or behavior scenarios, in particular, in the form of Gherkin scenarios.

2.2 User Stories

Initially, user stories were recorded on cards of small sizes. The card is not intended to collect all information about the requirement. The card must contain several sentences that reflect the essence of the requirement. The card usually indicates the identifier, name, text, acceptance criteria and its assessment (priority, risk, evaluation of the efforts, etc.). The user story should follow the following pattern:

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

Nevertheless, it is easy to make a lot of mistakes when writing a user story. In this connection, writing the user story text is one of the cornerstones of agile engineering requirements. Suppose it is necessary to describe the functionality that will allow the social network users to sell stickers sets that they can add to messages. In this case, the user has a standard free stickers set. Suppose the story was formulated as follows: "As a user, I want to add sticker sets from the paid collection, So that I can see new stickers in my sticker sets". In this user story, the type of user is not specified, there is no understanding of what problem the user solves, what is his motive. If the system users are divided into logged users and visitors, then it is probably a logged in user. And, most likely, the developer will immediately understand this and without losing time on figuring out the type of user will be able to implement the feature correctly. But if logged-in users are in turn subdivided (for example, there are users with a premium account that this service should already be included in the payment), the feature may be implemented incorrectly, or the developer will spend time clarifying out who the user is. In order to avoid mistakes in the first part of the user story, it is necessary to build a user roles model and accept an agreement that if this is not about any user, but about a certain group of users, this should be reflected in the user story. Further, the user story inaccurately defined user action and absolutely not indicated his motivation. This user story is better stated in the following wording: “As a logged-in user, I want to buy new stickers sets, So that I can decorate my messages with non-standard stickers”. Another important point when writing stories is the use of possessive pronouns, the omission of which can drastically change the meaning of the requirement.

Despite the huge popularity, the number of ways to assess the quality of user stories is small. Many existing approaches use the INVEST model proposed in 2003 by Bill Wake. According to this model, user history should be: Independent, Negotiable, Valuable, Estimable, Small, Testable [15].

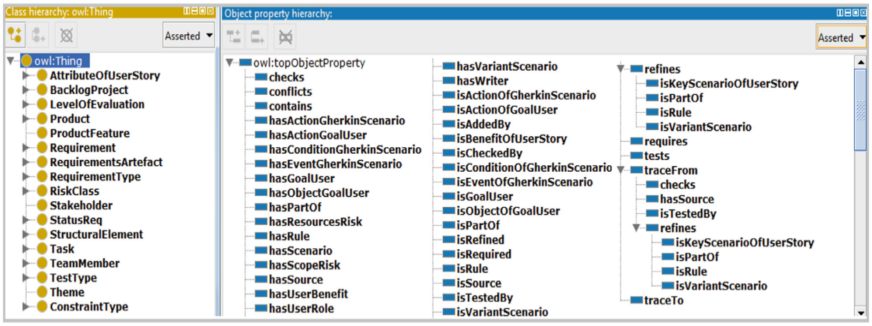
The aim of using the Independent criterion is to keep the number of dependencies on other user stories to a minimum. The presence of dependencies makes it difficult to prioritize and, accordingly, planning a sprint. The details of user stories should also be negotiable. According to the Negotiable criterion, the user story text should contain information only about what it needs to be implemented for. A typical example of a breach of Negotiable criterion is when a Product Owner tells a team how to implement a user story. According to the Valuable criterion, user stories should be valuable to the customer or end user, or both. According to the Estimable criterion, the user story must be clear to the development team so that the team can determine the necessary effort. If a team cannot estimate user story, then the user story is either too large, or ambiguously formulated, or the team does not have enough knowledge, for example, about some technology. In the first case, it is necessary to decompose the user story, in the second - to eliminate the ambiguity of the wording, in the third - to conduct research in order to obtain the necessary knowledge. When sprint planning, it is necessary that the user stories that are considered to be candidates for implementation have a small size so that several user stories can be planned for the sprint. Testable criterion means that each acceptance criteria must have a clear pass or fail result.

The development of a user stories list for the project can be preceded by the construction of the Feature tree [16]. Function trees are a good way to organize project information. The Product Owner begins building the Feature tree in Sprint 0. The start of the Feature tree can be generated based on information from the document “Vision and Scope”. Since all product features are usually undefined during sprint 0, the Feature tree is constantly evolving. Further, the items of the Feature tree are added to the Product backlog, where they are supplemented with user stories.

3 OWL Ontology for Agile Requirements Engineering

In this paper, it is proposed to use the OWL ontology system to support the requirements engineering. The first ontology contains knowledge of requirements engineering within the framework of an agile approach including knowledge about types of requirements. The second ontology is a model for identifying conflicting requirements. The third is a model that includes a software product feature tree, a user roles model and the connections between them.

In Fig. 2 shows the taxonomy of the upper level classes for ontology “Guide”, and also object properties reflecting the relations between ontology classes.



Class hierarchy: owl:Thing

Asserted

- owl:Thing
  - AttributeOfUserStory
  - BacklogProject
  - LevelOfEvaluation
  - Product
  - ProductFeature
  - Requirement
  - RequirementsArtefact
  - RequirementType
  - RiskClass
  - Stakeholder
  - StatusReq
  - StructuralElement
  - Task
  - TeamMember
  - TestType
  - Theme
  - ConstraintType

Object property hierarchy:

Asserted

- owl:topObjectProperty
  - checks
  - conflicts
  - contains
  - hasActionGherkinScenario
  - hasActionGoalUser
  - hasConditionGherkinScenario
  - hasEventGherkinScenario
  - hasGoalUser
  - hasObjectGoalUser
  - hasPartOf
  - hasResourcesRisk
  - hasRule
  - hasScenario
  - hasScopeRisk
  - hasSource
  - hasUserBenefit
  - hasUserRole
  - hasVariantScenario
  - hasWriter
  - isActionOfGherkinScenario
  - isActionOfGoalUser
  - isAddedBy
  - isBenefitOfUserStory
  - isCheckedBy
  - isConditionOfGherkinScenario
  - isEventOfGherkinScenario
  - isGoalUser
  - isObjectOfGoalUser
  - isPartOf
  - isRefined
  - isRequired
  - isRule
  - isSource
  - isTestedBy
  - isVariantScenario
  - refines
    - isKeyScenarioOfUserStory
    - isPartOf
    - isRule
    - isVariantScenario
  - requires
  - tests
  - traceFrom
    - checks
    - hasSource
    - isTestedBy
    - refines
      - isKeyScenarioOfUserStory
      - isPartOf
      - isRule
      - isVariantScenario
  - traceTo

Fig. 2. The taxonomy of the upper level classes for ontology “Guide” and object properties

The ontology “Guide” consists of classes corresponding to events, roles and artefacts of agile development. The instances (individuals) of the ontology classes are the software requirements and their artefacts, as well as information about the development team and stakeholders. Object properties reflect relations that can be established between individuals. For example, to specify the relationship “the requirement refines another requirement” the object properties “refines” are used. This object property, in turn, includes as subproperties that can be established between different classes (or individuals) of requirements artefacts which are also requirements by their nature (for example, the behavior scenario refines user story). Object property “traceFrom” is intended to define of bottom-up tracing links. For the object property “traceFrom” and its subproperties, inverse properties are given through the “Inverse Of” relationship. This allows the top-down tracing of the “traceTo” relationship. Object property “conflicts” enables specifying that the requirements conflict with each other. This can be done directly in this ontology or transferred from the additional ontology “Detection of conflicts in the requirements”. In Fig. 3 lists the objects properties for this ontology and their domains and ranges.

Individuals of the ontology “Detection of conflicts in the requirements” are elements extracted from the requirements text. The sentence that expresses a requirement, regardless of the technique used for recording requirements, can be divided into main parts: the subject, the action, and the object to which the action of the subject is directed. To identify conflicts between user stories need to extract from user story text the functional user role, the action and the object on which the action is directed. Object properties “sameAsActor”, “sameAsAction” and “sameAsObject” are set between instances of the corresponding classes if same name is used for the elements of two requirements or the full name in one and an abbreviation in the second. Object properties “isaActor” and “isaObject” are used to establish hierarchical relations. For example, a senior manager is a manager. Object properties “antonymsAction” and “antonymsObject” are set between instances of the corresponding classes if they have the opposite value (for example, “my comment about the product” and “someone else’s comment about the product”). Object properties “partOfAction” and “partOfObject” are set between instances of the respective classes if one is part of the other. Object properties “synonymsActor”, “synonymsAction” and “synonymsObject” are set if the values of the corresponding requirements elements have a synonymous value. In all other cases it is considered that the corresponding elements of the requirements are bound by object property "no-relationActor", "no-relationAction" or "no-relationObject".

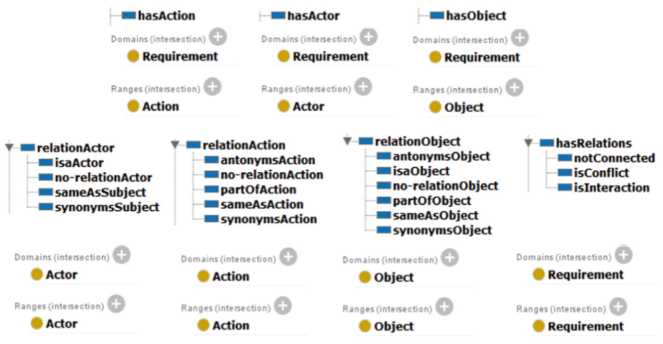


Fig. 3. Classes and object properties for ontology «Detection of conflicts in the requirements»

The following production rules are used to determine the type of relations between the two requirements:

If ObjectProperty_between_class_instances (Actor1, Actor2)
AND ObjectProperty_between_class_instances (Action1, Action2)
AND ObjectProperty_between_class_instances (Object1, Object2)
Then Object properties_between_class_instances (Requirement1, Requirement2).

Relations between actors, actions and objects can be established on the basis of information from the domain ontology as well as using linguistic ontologies, such as WordNet.

To illustrate the idea of the proposed approach to the formation of a software product domain model in the form of an OWL ontology consider a fragment of ontology for an online store (см. Fig. 4).

When building the software product domain ontology it is necessary to analyze the user roles (class "User") and also which software product sections users can work with (class "Office") and build a Feature tree (class "Features"). Further, it should be determined which sections can be used by certain users, and also indicate which features are associated with these sections. Instances of the class "Object" are objects that the user works with (for example, a sales report or a search string). Instances of the class "Action" are verbs that are used to describe user actions. Actions, respectively, can be divided into four operations: reading, adding, editing, deleting. In this example, users are divided into groups depending on the two properties "isLogin" and "isRegistered". The knowledge of which user office is owned is also used to assign a user to a class.

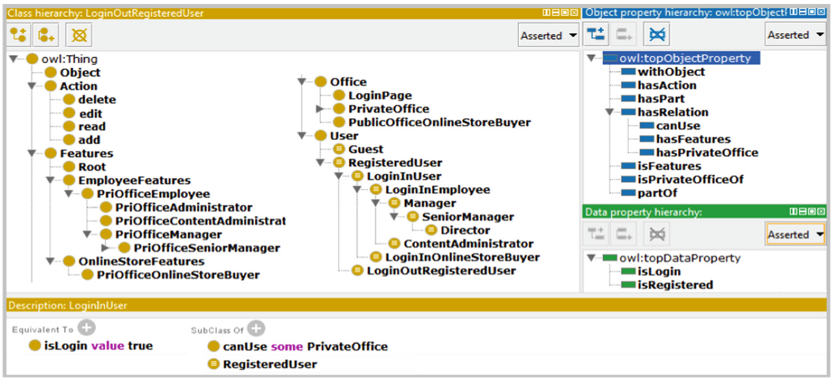


Fig. 4. Ontology «OnlineStore»

A list of the features available in office is indicated for each class of the corresponding office. Subclasses of personal offices automatically inherit features set for the classes in which they belong.

4 Conclusion and Future Work

The OWL ontology system containing three ontologies was developed based on the analysis of the results of ontology application in requirements engineering. The first ontology accumulates knowledge about the development of software products in a agile environment. The second one contains knowledge about the relations between the elements of sentence with a requirement (role, action, object). This allows analyzing pairs of requirements in order to identify conflicts. The third one contains knowledge of the software product domain.

We have developed a model through which enables solving typical problems for requirements engineering in agile software development. These include the formation and refinement of the Product backlog, requirements tracing and the conflicting requirements identification. The next stage of our research will include the development of the algorithm for prioritizing and re-prioritizing requirements based on the business value of the product backlog item, assessment of the efforts to requirements implement and risks, as well as the dependencies between product backlog items that are recorded in the ontology.

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

References

1. ISO/IEC. Software Engineering - Guide to the software engineering body of knowledge (SWEBOK). 2nd edn. ISO/IEC TR 19759 (2015)
2. Bhatia, M.P.S., Kumar, A., Beniwal R.: Ontologies for software engineering: past, present and future. Ind. J. Sci. Technol. **9**(9). http://www.indjst.org/index.php/indjst/article/view/71384/67982. Accessed 02 Feb 2019
3. Siegemund, K.: Contributions to ontology-driven requirements engineering: dissertation to obtain the academic degree doctoral engineer (Dr.-Ing.). Technischen Universität Dresden, Dresden (2014)
4. Valaski, J., Reinehr, S., Malucelli A.: Which roles ontologies play on software requirements engineering. In: International Conference on Software Engineering Research and Practice, pp. 24–30. CSREA Press (2016)
5. Castañeda, V., Ballejos, L., Caliusco, M.L., Galli, M.R.: The use of ontologies in requirements engineering. Glob. J. Res. Eng. **10**(6), 2–8 (2010)
6. Goknil, A., Kurtev, I., van den Berg, K.: A metamodeling approach for reasoning about requirements. In: Schieferdecker, I., Hartman, A. (eds.) ECMDA-FA 2008. LNCS, vol. 5095, pp. 310–325. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-69100-6_21
7. Assawamekin, N., Sunetnanta, T., Pluempitiwiriyawej, C.: Ontology-based multi perspective requirements traceability framework. Knowl. Inf. Syst. **25**(3), 493–522 (2010)
8. Sitthithanasakul, S., Choosri, N.: Using ontology to enhance requirement engineering in agile software process. In: 2016 10th International Conference on Software, Knowledge, Information Management and Applications, pp. 181–186. IEEE (2017)
9. Avdeenko, T., Murtazina, M.: Intelligent support of requirements management in agile environment. In: Borangiu, T., Trentesaux, D., Thomas, A., Cavalieri, S. (eds.) SOHOMA 2018. SCI, vol. 803, pp. 97–108. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-03003-2_7
10. Thamrongchote, C., Vatanawood, W.: Business process ontology for defining user story. In: 15th International Conference on Computer and Information Science. IEEE, Okayama (2016)
11. Murtazina, M.S., Avdeenko, T.V.: Ontology-based approach to the requirements engineering in agile environment. In: 2018 XIV International Scientific-Technical Conference on Actual Problems of Electronics Instrument Engineering (APEIE), pp. 496–501. IEEE, Novosibirsk (2018)
12. Adnan, M., Afzal, M.: Ontology based multiagent effort estimation system for scrum agile method. IEEE Access **5**, 25993–26005 (2017)
13. Agile Manifesto. https://agilemanifesto.org/. Accessed 02 Feb 2019
14. Rubin, K.S.: Essential Scrum: A Practical Guide to the Most Popular Agile Process. Addison-Wesley, Boston (2013)
15. Wake, B.: INVEST in Good Stories, and SMART Tasks. https://xp123.com/articles/invest-in-good-stories-and-smart-tasks/. Accessed 02 Feb 2019
16. Babar, M.A., Brown, A.W., Mistrik, I.: Agile Software Architecture: Aligning Agile Processes and Software Architectures, 1st edn. Elsevier, Waltham (2013)