Object-Oriented Approach for Requirements Specification

Maria Naumcheva[1,2]

[1]Innopolis University, 1 Universitetskaya St., Innopolis, Tatarstan republic, 420500, Russian Federation
[2]University of Toulouse / IRIT, Cr Rose Dieng-Kuntz, 31400 Toulouse, France

Abstract

Although the software engineering community knows well the deficiencies of natural language documentation, it remains the predominant way of software requirements specification. The desired properties of software requirements that are hard to be reached with natural language specifications are unambiguity and traceability. This issue has been solved through several approaches, yet still most software projects rely on natural language requirements.

This research aims at capitalizing on recent developments of programming language-based approaches to requirements with the purpose of devising a unified approach enriched with tools and methodology. This approach is based on the object-oriented technology as follows: the development process is seamless from the requirements specification to implementation and verification. An object-oriented language, Eiffel, is used as a requirements notation. It provides means of system modeling without using specific notation, which often becomes a barrier in formal methods adoption. The specification, design, implementation and tests are developed incrementally using the mechanisms of inheritance and refinement and relying on the notion of contracts.

1. Introduction

1.1. Problem

According to the IEEE international standard [1], a good software requirements specification should be correct, unambiguous, complete, consistent, ranked for importance and/or stability, verifiable, modifiable, and traceable. This research focuses on unambiguity and traceability of the requirements.

Ambiguity refers to possible different interpretations of the same requirement. Since natural language-based requirements specifications are a dominant practice in a software industry [2], eliminating this property is difficult to achieve because natural language (NL) is an innate source of ambiguity [3]. Naming the Pain in Requirements Engineering (NaPiRE) initiative [4] has also demonstrated statistical significance of the issue of "Underspecified requirements that are too abstract and allow for various interpretations"

In: J. Fischbach, N. Condori-Fernández, J. Doerr, M. Ruiz, J.-P. Steghöfer, L. Pasquale, A. Zisman, R. Guizzardi, J. Horkoff, A. Perini, A. Susi, M. Daneva, A. Herrmann, K. Schneider, P. Mennig, F. Dalpiaz, D. Dell'Anna, S. Kopczyńska, L. Montgomery, A. G. Darby, and P. Sawyer (eds.): Joint Proceedings of REFSQ-2022 Workshops, Doctoral Symposium, and Poster & Tools Track, Birmingham, UK, 21-03-2022, published at http://ceur-ws.org

m.naumcheva@innopolis.university (M. Naumcheva)

Researchers have addressed the NL-inherent ambiguity through different methods [5]. These methods suggest introducing certain level of formality: to constrain the natural language or to utilize formal methods and notations. Using formal methods implies the need to additionally train requirements engineers, as well as creates communication barriers between requirements engineers and software developers. As a consequence, such methods are not widely adopted since only 6% of respondents of the 2015 industrial survey [2] claimed formulating formal properties.

Requirements *traceability* is the ability to follow both the sources and consequences of requirements. Requirements traceability ensures that the impact of changes in requirements specification is easily localizable in the code, which significantly decreases the time and costs of assessing the impact of changes and implementing the changes. Traceability relations can also be used to assist verifying that the system meets its requirements. Maintaining requirements traceability links is often perceived as not a cost-efficient process, and many projects ignore this practice completely. At the same time, the projects with missing traceability links are vulnerable under change and evolution as it can be tremendously hard to identify all the system elements where the changes should be introduced.

This research develops a methodology of object-oriented approach for requirements specification. This approach relies on the modeling power of object-oriented programming language. Requirements are captured by contracts, which ensures their unambiguity. Applying object-oriented approach simplifies creating and maintaining traceability links as programming language artifacts can have direct links in IDE.

1.2. Related Work

Applying object-oriented concepts to requirements analysis and design attracted much attention in 1990s. The works of Rambaugh et al [6], Jacobson [7], and Booch [8] had much in common and were unified into Object-Oriented Analysis and Design (OOAD) [9] as a unified methodology, and Unified Modeling Language (UML) [10] as a unified notation and a set of graphical diagrams. OOAD suggests applying OO techniques to the initial requirements, produced at the earlier stages of the development process. This method relies on a set of diagrams to capture the software system details and to be able to communicate them to the project stakeholders.

Scenarios are considered to be the main technique for object-oriented requirements elicitation and specification. In addition to use cases, use cases 2.0 [11], use case stories, and user stories [12] were introduced. In the Use Case 2.0, Extreme Programming and Rational Unified Process approaches scenarios also play the key role in managing software development.

OOAD approach to requirements involves several notations: natural language, UML, and requirements specification language (such as OCL). This implies the need of transformations between notations and analysis models. Although some methods for automated conversion have been proposed [13], the process is prone to errors that can be difficult to catch due to switching from one notation to the other. In contrast, the suggested approach relies on seamless software development and provides means for formal properties modeling using the programming language as a notation.

1.3. Relevance

Requirements engineering remains a pain point: 45% of respondents are not satisfied with achievement of requirements engineering goals [2]. The suggested approach addresses important requirements problems (ambiguity, traceability), while being easy-to-grasp for requirements engineers as it relies on programming language mechanisms.

According to [4], practitioners agree with the following statements:

- The standardisation of requirements engineering improves the overall process quality
- Offering standardised document templates and tool support benefits the communication
- Offering standardised document templates increases the quality of the work products

Therefore, an approach that addresses current issues in requirements specification is beneficial to software development community in case if this approach is supported with methodology, templates and tools.

2. Proposed Solution

Object-oriented paradigm has been successful in software development yet its capabilities are underused for requirements elicitation, analysis and design. The research presented in this paper applies object-oriented approach to requirements. In this approach, object-oriented programming language is used to capture requirements with contracts, such as pre- and post-conditions, as well as class invariants. Software development is seamless as the unified notation, i.e. the programming language, is used throughout the entire development process. As a result, static verification can be applied to verify the correctness of the implementation against the requirements.

2.1. Object-oriented requirements

In the proposed approach, requirements become software: they are written in a programming language, and are stored in a version control system. The subsections 2.1.1 to 2.1.6 demonstrate the application of object-oriented concepts to requirements and what are the benefits of applying the same approach throughout all stages of development process.

2.1.1. Objects and classes

In object-oriented approach requirements are organised around physical or conceptual objects, rather than around procedures. The information about problem domain object types is carried in classes, the main units of abstraction. The features of the requirements classes are *deferred* and describe the operations, applicable to the objects of this type. The semantics of these operations is captured using preconditions, postconditions, and class invariants.

2.1.2. Abstraction

Abstraction refers to describing only relevant properties and dismissing irrelevant information. Abstraction is crucial in eliciting requirements since a requirements engineer must identify

the important aspects of the system and its environments and dismiss the rest [14]. Moreover, requirements specification and system design must avoid unnecessary implementation constraints.

In object-oriented approach the abstraction is induced by the following mechanisms:

- Defining the objects relevant to the system under development by assigning relevant features
- Defining the properties of objects and applicable operations using contracts.

The outcomes of requirements elicitation are often not abstract enough, and the task of a requirements engineer is to infer abstract requirements from concrete statements. For example, the outcome of an elicitation process can be a use case. The requirements engineer can extract from the use case abstract requirements that specify logical constraints on possible sequences of operations. These logical constraints can be captured by contracts and, thus, be enforced during system verification and validation.

2.1.3. Inheritance

Inheritance allows a class to incorporate the features of another class in addition to its own features. Inheritance can be viewed as a classification mechanism which is valuable in understanding the problem's domain.

Inheritance mechanism is indispensable for requirements specification. At the requirements phase the system requirements are captured in deferred classes. The effective classes, that will carry the system implementation, inherit from the deferred classes and must comply with their contracts. The induced inheritance graph captures traceability links.

2.1.4. Object-oriented Requirements notation

If the object-oriented approach is applied throughout the entire software development process, a programming language (the same one that will be used further for system implementation) becomes a notation for requirements. Unlike formal methods and UML, it does not require additional training for software engineers. Moreover, tools can support the automated translation of programming language specifications to a natural language. Related diagrams can also be derived automatically.

2.1.5. Seamless development process

Using programming language as a requirements notation facilitates the seamless development process. The specification, design, implementation and tests are developed incrementally using the mechanisms of inheritance and refinement. All software artifacts can be directly linked to each other in an IDE.

2.1.6. Contracts

According to Design by Contract, contracts are interface specifications for software classes and routines. Preconditions and postconditions are assertions associated with individual routines. Class invariants express the properties of all instances of a class.

Preconditions express the properties that must hold when calling the routine. Postconditions express the properties that must hold on the routine's exit. Class invariants apply to all contracts of a class.

Contracts serve as the mechanism of capturing requirements. Preconditions express the properties that must hold before invoking a feature. Postconditions express properties that must hold after invoking a feature. Class invariants express properties, which must be preserved by all features.

2.2. Requirements specification methodology

Requirements specification methodology is the key contribution of this research. It will address the following issues:

- How to specify environment properties, and how to turn them into requirements?
- What is the process of object-oriented requirements specification?
- What tool functionality should support the devised methodology?

2.3. Novelty

In model-driven development an intermediate modelling language is used for constructing the model of a system that can be further automatically translated to source code. Event-B [15], Petri-nets [16], and UML [17], [13] can serve as intermediate languages. However, formal methods are not widely adopted in industry [2]. Besides, in these approaches it is not possible to backtrack whether source code changes violate system's formal specification. Our research will address this limitation.

Applying seamless object-oriented approach to requirements was suggested by B. Meyer [18] and further developed by B. Meyer [19], A. Naumchev [20] and F. Galinier [21]. Nevertheless, the complete methodology of producing requirements specification has not been proposed, which makes the seamless approach unusable. Applying existing approaches to a case study revealed that they require methodological guidance to be applied to entire projects (rather than being used for specifying some particular properties).

The novelty of this research is in providing a methodology of producing object-oriented requirements and supporting this methodology with a tool prototype. As the approach diverges from the current practice, its usability plays vital role in its adoption.

3. Research Method

We aim to answer the following research questions:

- What is the proper role of natural language in requirements specification?
- Do object-oriented programming languages have the expressive power to model functional and non-functional requirements?
- What are the means of ensuring requirements unambiguity in OO approach?
- What are the means of ensuring requirements traceability in OO approach?
- What is the role of scenarios in the requirements specification methodology?

- What is the functionality of a tool that should support the OO approach?

This research consists of three stages.

Initially a study of the state of the art is conducted as well as an investigation of the approaches to requirements specification that address the issue of requirements ambiguity and a state of practice of their application in industry. Further a study of the methodology of commonly used approaches to requirements and their tool support is carried out.

The next stage is developing a methodology of applying object-oriented approach to requirements. We use a real-world Roborace case study to develop a methodology in the process of producing its requirements specification. Producing requirements specification, together with the output of the state-of-practice research on tools supporting requirements specification, will result in proposing functionality of a tool supporting the approach. The initial tool functionality will be provided early during methodology development. Further the tool functionality will be refined in the process of Roborace case study exploration. The methodology will also be supported with templates, publicly available on GitHub.

The final stage is validating the proposed methodology. For this purpose we will apply it to a new case study and demonstrate the soundness of the generated specification.

4. Progress

The initial phase of this research involved studying the literature and exploring state of the art and state of practice. Based on this study, we concluded that requirements methods addressing the issue of requirements ambiguity assume introducing certain level of formality whereas software development community remains reluctant to introducing formal methods. Only 40% of software development projects perform traceability management between requirements and other software artifacts [2].

The next step was to explore a case study which we use as a testbed for the methodology under development. The initial requirements specification for Roborace case study was produced. Based on this specification, we suggested an approach that unifies scenarios and object-oriented requirements specification.

Currently we are working on the methodology of deriving requirements from environment properties and specifying the requirements in an object-oriented way. Further we will formulate the methodology and tool functionality, which will be followed with the validation study.

5. Conclusion

In this paper we have presented a research that aims to provide a methodology of producing unambiguous software requirements. The methodology relies on an object-oriented approach to requirements. In this approach a programming language is used as the notation for requirements formalization. The resulting software development process is seamless from requirements to implementation, verification and validation, which ensures traceability between requirements and other software artifacts and facilitates system verification and validation.

References

[1] I. C. S. S. E. S. Committee, I.-S. S. Board, IEEE Recommended Practice for Software Requirements Specifications, volume 830, IEEE, 1998.

[2] S. A. Fricker, R. Grau, A. Zwingli, Requirements engineering: best practice, in: Requirements Engineering for Digital Health, Springer, 2015, pp. 25–46.

[3] B. Meyer, On formalism in specifications, IEEE Software 2 (1985) 6–26.

[4] D. M. Fernández, S. Wagner, Naming the pain in requirements engineering: A design for a global family of surveys and first results from Germany, Information and Software Technology 57 (2015) 616–643.

[5] J.-M. Bruel, et al., The role of formalism in system requirements, ACM Computing Surveys (CSUR) 54 (2021) 1–36.

[6] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. E. Lorensen, et al., Object-oriented modeling and design, volume 199, Prentice-hall Englewood Cliffs, NJ, 1991.

[7] I. Jacobson, M. Christerson, P. Jonsson, G. Övergaard, Object-oriented software engineering - a use case driven approach, Addison-Wesley, Boston MA, 1992.

[8] J. Rumbaugh, I. Jacobson, G. Booch, The Unified Modeling Language Reference Manual, Object Technology Series, 2 ed., Addison-Wesley, Boston, MA, 2004.

[9] G. Booch, et al., Object-oriented analysis and design with applications, ACM SIGSOFT software engineering notes 33 (2008) 29–29.

[10] O. OMG, Unified modeling language version 2.5.1, Object Management Group (2017).

[11] I. Jacobson, I. Spence, K. Bittner, USE-CASE 2.0 The Guide to Succeeding with Use Cases, Ivar Jacobson International SA., Alexandria, Virginia, 2011.

[12] K. Beck, Extreme programming explained: embrace change, addison-wesley professional, 2000.

[13] T. Yue, et al., atoucan: an automated framework to derive uml analysis models from use case models, ACM Trans. on Soft. Eng. and Methodology (TOSEM) 24 (2015) 1–52.

[14] J. Kramer, Is abstraction the key to computing?, Communications of the ACM 50 (2007).

[15] V. Rivera, N. Catano, T. Wahls, C. Rueda, Code generation for event-b, International Journal on Software Tools for Technology Transfer 19 (2017) 31–52.

[16] S. Philippi, Automatic code generation from high-level petri-nets for model driven systems engineering, Journal of Systems and Software 79 (2006) 1444–1455.

[17] T. Yue, et al., Facilitating the transition from use case models to analysis models: Approach and experiments, ACM Trans. on Soft. Eng. and Methodology (TOSEM) 22 (2013) 1–38.

[18] B. Meyer, Object-oriented software construction, volume 2, Prentice hall Englewood Cliffs, 1997.

[19] B. Meyer, Multirequirements, in: Modelling and Quality in Requirements Engineering (Martin Glintz Festscrhift), Verl.-Haus Monsenstein u. Vannerdat, 2013.

[20] A. Naumchev, Seamless Object-Oriented Requirements, in: 2019 International Multi-Conference on Eng., Computer and Inform. Sciences (SIBIRCON), IEEE, 2019, pp. 743–748.

[21] F. Galinier, Seamless development of complex systems: a multirequirements approach, Ph.D. thesis, Université de Toulouse, Université Toulouse III-Paul Sabatier, 2021.