Information and Software Technology 56 (2014) 321–334

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

Model-based requirements verification method: Conclusions from two controlled experiments

Daniel Aceituna [a,1], Gursimran Walia [a,2], Hyunsook Do [a,3], Seok-Won Lee [b,*]

[a] Department of Computer Science, North Dakota State University, IACC 258, 2740, P.O. Box 6050, Fargo, ND 58108-6050, United States
[b] Department of Software Convergence Technology at Ajou University, San 5 Woncheon-dong, Youngtong-gu, Suwon-si, Gyeonggi-do 443-749, Republic of Korea

ARTICLE INFO

Article history:
Received 15 January 2013
Received in revised form 6 November 2013
Accepted 11 November 2013
Available online 22 November 2013

Keywords:
Requirements verification
Model-based verification
NLtoSTD
Fault checklist
Controlled experiments

ABSTRACT

Context: Requirements engineering is one of the most important and critical phases in the software development life cycle, and should be carefully performed to build high quality and reliable software. However, requirements are typically gathered through various sources and are represented in natural language (NL), making requirements engineering a difficult, fault prone, and a challenging task.
Objective: To ensure high-quality software, we need effective requirements verification methods that can clearly handle and address inherently ambiguous nature of NL specifications. The objective of this paper is to propose a method that can address the challenges with NL requirements verification and to evaluate our proposed method through controlled experiments.
Method: We propose a model-based requirements verification method, called NLtoSTD, which transforms NL requirements into a State Transition Diagram (STD) that can help to detect and to eliminate ambiguities and incompleteness. The paper describes the NLtoSTD method to detect requirement faults, thereby improving the quality of the requirements. To evaluate the NLtoSTD method, we conducted two controlled experiments at North Dakota State University in which the participants employed the NLtoSTD method and a traditional fault checklist during the inspection of requirement documents to identify the ambiguities and incompleteness of the requirements.
Results: Two experiment results show that the NLtoSTD method can be more effective in exposing the missing functionality and, in some cases, more ambiguous information than the fault-checklist method. Our experiments also revealed areas of improvement that benefit the method's applicability in the future.
Conclusion: We presented a new approach, NLtoSTD, to verify requirements documents and two controlled experiments assessing our approach. The results are promising and have motivated the refinement of the NLtoSTD method and future empirical evaluation.

1. Introduction

Requirements verification is a process of determining whether requirements specifications capture the desired features of the system being built correctly. This process can be difficult because, typically, requirements are gathered through various sources and are represented in natural language (NL) as a means of communication between different stakeholders (i.e., both technical and non-technical). Requirements written in NL are prone to errors due to the inherent imprecision, ambiguity, and vagueness of natural language. Evidence suggests that if left undetected, these requirement errors can cause major re-work during the later stages of software development (i.e., during the implementation and testing phases). Furthermore, finding and fixing problems earlier rather than later is cheaper, and less expensive [7,36]. To ensure high-quality software, successful organizations focus on identifying and correcting problems in the software artifacts developed during the early stages of the software-development lifecycle.

To ensure requirements quality, to date, researchers have developed various verification methods (Section 7 summarizes the related work.) for detecting and removing the early lifecycle faults (i.e., mistakes recorded in the requirements and design documents [6,8,10,16]) and have validated the methods through controlled experiments and case studies (e.g., [9,11,21,26]). In particular, *software inspections*, in which a team of skilled individuals review a software work-product (e.g., a requirements document or a design document) to identify faults, is an effective verification method. Software inspections have been widely used to help developers identify different types of early lifecycle faults. To aid developers

* Corresponding author. Tel.: +82 31 219 3548; fax: +82 31 219 1621.
E-mail addresses: daniel.aceituna@ndsu.edu (D. Aceituna), gursimran.walia@ndsu.edu (G. Walia), hyunsook.do@ndsu.edu (H. Do), leesw@ajou.ac.kr (S.-W. Lee).
[1] Tel.: +1 701 231 8562; fax: +1 701 231 8255.
[2] Tel.: +1 701 231 8185; fax: +1 701 231 8255.
[3] Tel.: +1 701 231 5856; fax: +1 701 231 8255.

0950-5849/$ - see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.infsof.2013.11.004

with detecting a larger number of faults during an inspection, researchers have developed variants of Fagan's inspection [16] that range from an ad hoc inspection to a simple, fault-based checklist to a more detailed, step wise abstraction of the artifact.

However, even when faithfully applying these methods, it is estimated that 40–50% of development effort is still spent fixing problems that should have been corrected early in the lifecycle [7]. Much of this rework is the result of the fact that inspection methods rely on the inspectors' abilities to understand the requirements, and often, their interpretations are different from what the requirement developers intended. Because of the flexibility and inherently ambiguous nature of NL specifications, inspectors can have different interpretations of the same requirements without noticing the ambiguities and inconsistencies. Further, as the size of the requirement document grows, this tendency also increases.

Model-based approaches [2,19,20,24] can detect such faults more easily because, when the requirements are modeled or checked with formal methods, the properties, such as inconsistencies and ambiguities, are clearly addressed and handled. For this reason, many researchers have utilized model-based approaches for verifying NL specifications.

While model-based approaches provide a systematic way to identify inconsistent and incomplete requirements, building models often requires NL translation, and this translation process can be highly subjective because stakeholders can interpret NL requirements differently [5,17]. An erroneous translation of NL requirements can result in the wrong model and, thus, can eventually produce software that stakeholders do not want. To alleviate the problem with the erroneous translation, several researchers have proposed modeling techniques using an automated NL translation approach [4,13,15,22]. Automation can certainly reduce human errors and improve the translation process, but complete and error-free automation of this process is not possible because, often, NL requirements can be interpreted in multiple ways; therefore, human judgment can inevitably lead to various correct and sensible interpretations.

To address these problems with manual inspection methods and model-based methods, we propose a new method that translates NL requirements into a State Transition Diagram (STD) in an incremental manner (hereafter referred to as NLtoSTD) and allows requirement engineers and other stakeholders to participate in the translation process. This approach can correct and refine requirements during the translation process by identifying ambiguities and incompleteness in the NL requirements.

The NLtoSTD method provides a means of exposing faults in a set of NL requirements while transforming the requirements into an STD. While the requirement faults have also been explored by other approaches [25,30], our method differs from the existing techniques in that the direct mapping from NL to an STD model is preserved in the translation process. Each NL requirement becomes a segment of the STD, resulting in a direct mapping between the requirements and the STD. This means that any adjustments or changes made to the STD can be directly traced back to the requirements, and vice versa. (A detailed example of this process is shown in Section 2.2.)

To investigate the feasibility and applicability of our approach, we designed and conducted two controlled experiments which evaluated the user's fault-detection ability during the translation of NL requirements (contained in different requirement documents) into the building blocks (or segments) of the STD. These experiments validated the use of translating NL requirements into STD segments as an effective requirements verification method. Our results showed that the proposed method can be improved to make it more effective in exposing the missing functionality and ambiguous information in NL requirements when compared to a fault-checklist method.

1.1. Our contribution

This paper is substantially expanded from the previous published results [37], and the new contributions are highlighted as follows:

(1) *Using more extensive universe of data*: This paper reports the results from a second experiment that used the improved NLtoSTD-BB method (based on the lessons learned from the first experiment) and was designed to address the validity threats unaddressed in the first experiment. Further, we also included new data analysis from the first experiment that has not been reported elsewhere.
(2) *Stronger findings*: We integrated the results of the Data Sets (of different subjects and requirement artifacts) from both the experiments to draw more general conclusions and compared findings against previous findings.
(3) *Additional insights and Future research directions*: We also discuss the further improvement of the latest version of the NLtoSTD-BB (that was used in the second experiment) to highlight the strengths and limitations of the NLtoSTD-BB method during the inspection of NL requirement documents.
(4) Through the experimental results, we were able to highlight the promise of using NLtoSTD-BB as a requirements verification method that is more effective in detecting requirements faults than the traditional inspection method. Our method preserves the direct mapping from requirements to the model during the translation process, which supports traceability between two software artifacts (requirements and the model). Our method allows both technical and non-technical stakeholders participate in the requirement verification process.

The rest of the paper is organized as follows. Section 2 describes our NLtoSTD approach in detail, and Section 3 provides the experimental framework used to evaluate our research approach. Sections 4 and 5 present our two experiments, including the design, threats to validity, data and analysis, and result interpretation. Section 6 discusses the results we observed from the two experiments and the practical implications. Section 7 discusses related work. Finally, Section 8 presents conclusions and discusses possible future work.

2. NLtoSTD method

This section explains the NLtoSTD method and describes the application of the NLtoSTD-BB on a set of example NL requirements for detecting faults.

2.1. Basic concepts underlying the NLtoSTD method

The rationale behind developing the NLtoSTD method was to be able to translate NL requirements into a formalized form, which can facilitate exposing incompleteness and ambiguities in the original NL requirements. An STD is a formal description of the system-to-be behavior, and that it can be decomposed into building blocks (BBs) that make up the STD. Assume that a complete and unambiguous STD exists, that means each BB (that makes up the STD) is also complete and unambiguous. This indicates that complete and unambiguous STD-BBs represent a set of NL requirements (that were translated into BBs) that do not contain faults.

The NLtoSTD method consists of two steps. The first step includes the translation of NL requirements into STD-BBs (thereby referred to as NLtoSTD-BB) and the second step includes the construction of an STD using the STD-BBs (referred as STD-BBtoSTD).