Information and Software Technology 56 (2014) 40–53

Contents lists available at SciVerse ScienceDirect

Information and Software Technology

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

Investigating dependencies in software requirements for change propagation analysis

He Zhang [a,*], Juan Li [c], Liming Zhu [b], Ross Jeffery [b], Yan Liu [d], Qing Wang [c], Mingshu Li [c]

[a] State Key Laboratory of Novel Software Technology, Software Institute, Nanjing University, Jiangsu, China
[b] NICTA, University of New South Wales, Australia
[c] Institute of Software, Chinese Academy of Sciences, China
[d] Faculty of Engineering and Computer Science, Concordia University, Canada

ARTICLE INFO

Article history:
Available online 20 July 2013

Keywords:
Requirement dependency
Requirement traceability
Requirement relationship
Change propagation
Impact analysis
Case study

ABSTRACT

Context: The dependencies between individual requirements have an important influence on software engineering activities e.g., project planning, architecture design, and change impact analysis. Although dozens of requirement dependency types were suggested in the literature from different points of interest, there still lacks an evaluation of the applicability of these dependency types in requirements engineering.

Objective: Understanding the effect of these requirement dependencies to software engineering activities is useful but not trivial. In this study, we aimed to first investigate whether the existing dependency types are useful in practise, in particular for change propagation analysis, and then suggest improvements for dependency classification and definition.

Method: We conducted a case study that evaluated the usefulness and applicability of two well-known generic dependency models covering 25 dependency types. The case study was conducted in a real-world industry project with three participants who offered different perspectives.

Results: Our initial evaluation found that there exist a number of overlapping and/or ambiguous dependency types among the current models; five dependency types are particularly useful in change propagation analysis; and practitioners with different backgrounds possess various viewpoints on change propagation. To improve the state-of-the-art, a new dependency model is proposed to tackle the problems identified from the case study and the related literature. The new model classifies dependencies into *intrinsic* and *additional* dependencies on the top level, and suggests nine dependency types with precise definitions as its initial set.

Conclusions: Our case study provides insights into requirement dependencies and their effects on change propagation analysis for both research and practise. The resulting new dependency model needs further evaluation and improvement.

1. Introduction

In developing software intensive systems, most individual requirements relate to and affect each other in complex manners [26]. Various and complex dependencies among requirements increase the difficulties in understanding the requirements and influence many software engineering activities such as architecture design [18], product release planning [6] and change impact analysis [27]. Requirement dependencies are also important inputs for component selection [14] and web service composition [31].

In recent volatile environment, software systems must evolve to adapt themselves to the rapid changes of stakeholders' needs, technologies and business environments [20]. To control the risks brought by software evolution, it is vital to analyse change propagation in order to determine what other parts of a software may be affected if a change is made. Evolution of software systems is mostly studied at the level of code and design with a focus on code reengineering/migration, architectural evolution and software refactoring [1,34]. However, it is also necessary to analyse change propagation earlier at the requirements level. This kind of change propagation analysis can provide important change-related information from a business point of view. Requirements-level change propagation analysis has also been regarded as one important area in software evolution research [13].

Requirements dependency is the relationship between requirements and acts as the basis for change propagation analysis.

* Corresponding author. Tel.: +86 25 83621369; fax: +86 25 83621370.
E-mail addresses: dr.hezhang@gmail.com (H. Zhang), lijuan@itechs.iscas.ac.cn (J. Li), liming.zhu@nicta.com.au (L. Zhu), ross.jeffery@nicta.com.au (R. Jeffery), yan.liu@concordia.ca (Y. Liu), wq@itechs.iscas.ac.cn (Q. Wang), mingshu@itechs.iscas.ac.cn (M. Li).

0950-5849/$ - see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.infsof.2013.07.001

Dozens of dependency types have been proposed to reflect the complex relationships between requirements at both structural and semantic levels. These dependency types have different levels of abstraction and are used in various aspects of project management. Pohl [24], Dahlstedt and Persson [9] proposed the dependency types respectively based on literature survey in the areas of requirements engineering. Karlsson et al. introduced the dependency types to prioritise requirements [17]. Carlshamre et al. presented a study of requirements dependencies in release planning [7].

However, there still lacks an evaluation of the applicability of these dependency types in a real-world project and their effectiveness in change propagation analysis. This limits the wide use of these dependency types for both dependency identification and change propagation analysis. In practise, researchers and practitioners usually choose from these dependency types based on their own experiences and ignore other dependencies that may propagate changes. As a result, the accuracy of the change propagation result may decrease. In one of our early studies [23] we chose three dependency types arbitrarily from Dahlstedt's dependency model [9] to estimate the change impact at the code level. This choice ignored other dependency types in Dahlstedt's dependency model that may have propagated changes and subsequently influenced our change propagation estimation. In addition, some change impact analysis research merely focuses on change propagation in a specific requirements model, such as use case map model [15], which severely constrains the range of dependency types and causes change propagation analysis incomplete.

Given the very limited evaluation endeavour on requirements dependency, our research aims to empirically investigate the usefulness and applicability of existing dependency models (types), and further propose the improvements that are based upon the evaluation results to effectively facilitate dependency identification and change propagation analysis. To achieve the goals, we conducted a case study with three participants to evaluate the usefulness and applicability of existing dependency types in a real-world industry project. The evaluation objects are 25 dependency types defined in Dahlstedt's dependency model (*D*-model) [9] and Pohl's dependency model (*P*-model) [24], which are the two most adopted dependency models in software requirements. This case study is aimed to answer four research questions:

RQ1 *What dependency types are (not) used to identify relationships between requirements and why?*
RQ2 *What dependency types are (not) used in change propagation analysis and why?*
RQ3 *How effective/ineffective are dependency types used?*
RQ4 *What are the main factors affecting the discovery of dependencies?*

Corresponding with these research questions, the evaluation presented in this article has found that: (1) seven dependency types in the *P*-model and three dependency types in the *D*-model were deemed applicable by the practitioners to describe relationships between requirements; (2) five dependency types can indicate change propagation particularly well, but their definitions need to be clarified; (3) dependency types are helpful to find more dependencies, but some dependency types have ambiguous definitions or overlap with each other which increases the difficulty of use; and (4) four main factors affecting the discovery of dependencies. We also find participants with various backgrounds have different viewpoints on change propagation. Change impact analysis should involve a wide range of stakeholders including project managers, requirements engineers, designers and developers. Moreover, we provide a group of specific dependency types for change propagation analysis based on our empirical findings.

Based upon our empirical evaluation of existing dependency models and the critique of current dependency types, we further propose a new dependency model by refining, integrating and extending the previous ones. The new dependency model classifies dependencies into *intrinsic* dependencies and *additional* dependencies on the top level. The former reflects essential dependencies existing among requirements, which are more likely to impact many various software engineering activities; the latter represent relationships imported by certain software engineering tasks. The initial set of the new model includes nine generic dependency types.

This article is the extended version of the conference paper published in EASE 2012 [22]. Compared to the original version, the major extensions in this article are the detailed description of the case study design, execution, and data analysis (Section 3 and 4), as well as the presentation of a new dependency model (Section 5) based on the findings from the case study. In addition, this article also include numerous minor updates, corrections, and enhancements to the conference paper.

The rest of this article is structured as follows. Section 2 introduces the background and the related work about requirements traceability, requirement dependency, and change propagation analysis, and also briefly compares the existing dependency types whose problems motivated our research. Section 3 describes the context and method of the case study for evaluating the existing dependency types. The results and analysis of the case study are reported in Section 4 for answering the research questions in terms of usefulness and applicability. Section 5 proposes a new dependency model and redefines a set of common dependency types based on our empirical findings. Section 6 discusses the dependency types for change propagation analysis in particular as well as the limitations of our study at this stage. We conclude our research with suggestions for further work in Section 7.

2. Background and motivations

2.1. Dependency in requirements engineering

In order to analyse the impact of a proposed software change, it is vital to determine which parts of the software system may be affected by the change and ascertain their possible risks [5]. Change propagation means that a change can impact on not only source code, but also other software artifacts, such as requirements, design and test cases [4]. Bohner proposed an impact analysis process [5] which examines the change requests to identify the Starting Impact Set (SIS) of software artifacts that could be affected by the required change. Artifacts in the SIS are then analyzed to identify other artifacts anticipated to be affected. The newly identified artifacts are incorporated into SIS to form the Candidate Impact Set (CIS). On the other hand, an Actual Impact Set (AIS) consists of the set of artifacts actually modified after the change is implemented. The goal of the impact analysis process is to estimate a CIS that is as close as possible to the AIS. Along this topic, Fasolino and Visaggio [12] showed how CIS can be determined step by step starting from the highest-level documentation (e.g., requirements specification) affected by the changes down to the source code to be changed. This top-down analysis approach can be efficiently supported through traceability information [10].

Traceability research is gaining an increasing attention in many areas such as requirements engineering and model driven architecture [29]. A traceability link is "*any relationship that exists between artefacts involved in the software engineering life cycle*" [2]. It includes not only the forward and backward links between artefacts (e.g., requirements and architecture, requirements and code), but also links between items within a software development