Chapter 5

Enhanced Requirement Traceability Link Using Developer’s Updated Activity

Raviraj Pandian
GSSS Institute of Engineering and Technology for Women, India

Anandha Kumar D.
Kalaignar Karunanidhi Institute of Technology, India

ABSTRACT

In the software development cycle, the requirement traceability link is one of the important factors. Information retrieval techniques typically produce links with low precision and/or recall because, due to their very nature, they depend on the textual similarity between requirements and source code. The developers may not evolve requirements in synchronization with source code. But, they frequently update other sources of information including CVS/SVN repositories, bug-tracking systems, mailing lists, forums, and blogs. These sources of information will be used to build improved requirement traceability-recovery approaches. After the source code was developed, the developer compiles the source code and commits the source code into versions. Then admin needs to compare the templates and versions based on similarity measurement. Based on textual comparison, the similarity is calculated. The admin can do the same process for the forum requirements.
INTRODUCTION

Researchers mine software repositories to get a better understanding of the complex and constantly evolving social-technical interactions that take place during a long software development project. For example, researchers use historical data to predict the most problematic modules requiring more attention from developers or to advise which developers are the most knowledgeable about a certain module. Requirement traceability is defined as “the ability to describe and follow the life of a requirement, in both a forwards and backwards direction. Traceability links between requirements of a system and its source code are helpful in reducing system comprehension effort (Mader, O. Gotel, and I. Philippow, 2008). They are also essential to ensure that a system’s source code is consistent with its requirements and that all and only the specified requirements have been implemented by developers. Yet, during software maintenance and evolution, as developers add, remove, or modify features, requirement traceability links become obsolete because developers do not/cannot devote effort to update them. Yet, recovering later these traceability links is a daunting and costly task for developers. Consequently, the literature proposed methods, techniques, and tools to recover these traceability links semi-automatically or automatically.

Importance of Software Requirement Traceability

Preliminary to any software evolution task, a developer must comprehend the project landscape, in particular, the system architecture, design, implementation, and the relations between the various artifacts using any available documentation. Program comprehension occurs in a bottom-up manner, a top-down manner, or some combination thereof. Developers use different types of knowledge during program comprehension, ranging from domain-specific knowledge to general programming knowledge (Mader, O. Gotel, and I. Philippow, 2008). Traceability links between source code and sections of the documentation, e.g., requirements, aid both top-down and bottom-up comprehension. Requirement traceability is defined as the ability to describe and follow the life of a requirement, in both a forwards and backwards direction (i.e., from its origins, through its development and specification, to its subsequent deployment and use, and through all periods of on-going refinement and iteration in any of these phases)”. Traceability links between the requirements of a system and its source code are helpful in reducing comprehension effort. This traceability information also helps in software maintenance and evolution tasks (X. Zou, R. Settimi, and J. Cleland-Huang 2010). For example, once a developer has traceability links, she can easily trace what software artifacts must be modified to implement a new requirement. Traceability links are also essential to ensure that a
Component-Based Decision Trees: Empirical Testing on Data Sets of Account Holders in the Montenegrin Capital Market
[www.igi-global.com/article/component-based-decision-trees/133602?camid=4v1a](www.igi-global.com/article/component-based-decision-trees/133602?camid=4v1a)

Engineering Management: The Evolution, Conceptual Model, and Social Responsibility of an Emerging Discipline
[www.igi-global.com/article/engineering-management/212675?camid=4v1a](www.igi-global.com/article/engineering-management/212675?camid=4v1a)