Use of Ontology in Identifying Missing Artefact Links

Foziah Gazzawe
Department of Computer Science
Loughborough
United Kingdom
f.gazzawe@lboro.ac.uk

Russell Lock
Department of Computer Science
Loughborough
United Kingdom
R.Lock@lboro.ac.uk

Christian Dawson
Department of Computer Science
Loughborough
United Kingdom
C.W.Dawson1@lboro.ac.uk

ABSTRACT

The techniques of requirement traceability have evolved over recent years. However, as much as they have contributed to the software engineering field, significant ambiguity remains in many software engineering processes. This paper reports on an investigation of requirement traceability artefacts, stakeholders, and SDLC development models. Data were collected to gather evidence of artefacts and their properties from previous studies. The aim was to find the missing link between artefacts and their relationship to one another, the stakeholders, and SDLC models. This paper undertakes the first phase of the main research project, which aims to develop a framework for guiding software developers to actively manage traceability. After inquiring into and examining previous research on this topic, the links between artefacts and their functions were identified. The analysis resulted in the development of a new model for requirement traceability, defined in the form of an ontology portraying the contributively relations between software artefacts using common properties with the aid of Protégé Software. This study thus provides an important insight into the future of the requirement artefacts relation, and thereby lays an important foundation towards increasing our understanding of their potential and limitations.

KEYWORDS

Requirements Traceability; Requirement Artefacts; Artefacts Link; Mapping the Requirement Artefacts

ACM Reference format:

F. Gazzawe, R. Lock, and C. Dawson. 2018. 7th International Conference on Software and Computer Applications (ICSCA 2018) -- Ei Compendex, Scopus and ISI, *Kuantan, Malaysia, Feb 2018,* (ICSCA'7), 5 pages.
DOI:

1. INTRODUCTION

Requirement traceability is one of the most important aspects in determining the success of a project. It is through traceability that the correlation that exists between software, artefacts and their link to requirements is identified and managed effectively in software development. The association between these artefacts and the way they depend on each other is understood through the process of software development called Requirements Traceability (Requirements.com, 2017). Previous research on requirement traceability has not sufficiently defined the full range of artefacts potentially involved in traceability, nor has it considered the relationships between them. It can be argued that this missing link frequently leads to mistakes and misunderstandings when developing systems, thereby resulting in failure, as issues in software planning, design and implementation cannot be detected efficiently. This issue necessitated the development of traceability tools, however current tools are limited in their scope. Requirement traceability artefacts link to each other in many different ways.

According to De Lucia et al. (2011), when it comes to the management and control of the whole evolution of a software system, requirement traceability is especially useful. One aspect of this is in the identification of missing requirements in a software development process. Most importantly, traceability involves identification of the links between the requirement traceability artefacts and relating them to each other. Therefore, the relationships existing between artefacts can be defined as links. One example of a created link is when an artefact is embedded into a document to facilitate traceability (Levy, 2010).

Examples of artefacts available and commonly encountered include specifications, diagrams, and code. There are different classifications of artefacts based on architecture, technology-based decisions, modelling language, technical orientation and user-oriented elements. Some types of artefacts include container, generic document, individual element, user-oriented element, solution models, concrete models, abstract models, modeling language, and architectural models. These are different types of artefacts all aimed at assisting in software development and identification of requirements in software development (Olga, 2015).

The aim of identifying missing links between requirement is to ensure that the quality of a software solution is enhanced, resulting in better functionality and understanding of the system. The research objective is to determine whether a common model for requirement artefacts can be constructed. In the next section, software requirements, development, and traceability and artefacts will be defined and explained.

1.1 Literature Review

Traceability is achieved through links as they enable checking and verification of whether requirement information is traced through project documentation. A traceability link is a representation of the relation between two objects. Consistent traceability in a given project enables developers to trace back the

related information on software development (SDL Forum & Khendek, 2013). However, a missing link in artefact requirement exists which has led to poor software development in terms of requirement traceability. This paper argues that the use of ontology would be an effective approach in improving the visibility of traceability links.

2. SOFTWARE REQUIREMENTS, DEVELOPMENT, TRACEABILITY AND ARTEFACTS

2.1 Software Requirements

According to the IEEE standard 610.12-1990, IEEE (1990), a software requirement is defined as "A condition or capability needed by a user to solve a problem or achieve an objective" (Requirements.com, 2017). As regards the importance of gathering requirements, requirements may be seen as providing the necessary basis for software development (Young, 2003). Not defining requirements increases the risk of a project failing because development teams cannot communicate effectively, and consequently, they fail to explain what needs to be built, which can lead to catastrophic consequences. Requirement artefacts on the other hand, can be described as deliverables that are achieved in a software process, or those deliverables that are categories in certain tasks in software development. Their incorporation adds value to a certain role that the software is meant to play (Miles et al. 2012).

2.2 Software Development Lifecycle (SDLC)

It is important to have an understanding of different software development lifecycle models. These can be categorized at a coarse-grained level into two major contrasting types, namely Waterfall and Agile Models. Formal requirements documentation features more predominantly in the Waterfall Model. Importantly, it is in this and other similar models in which development commences with ascertaining requirements before starting work on developing the system architecture, making a detailed software design and undertaking the task of programming. Although requirements are gathered at an early stage in the waterfall model, requirement artefacts is a term that is present throughout the lifecycle in the form of design documentation and so on.

These traditional requirements-based models stand in contrast with models that guide the development of what is called agile software development methods in which not all requirements are gathered upfront or given the same level of importance, and an iterative model is followed. In this way, the development of the software and inclusion of functionality is incremental. The idea behind agile development is to strike a balance between ensuring satisfactory software quality, timely delivery and a reasonable cost (Liu et al., 2013).

The general understanding is that detailed planning helps to reduce uncertainty, provides clarity in project goals and objectives, and overall, it improves efficiency (Wysocki, 2011). Although planning before coding does consume extra time, the rationale for doing so is that this planning pays off, as the initial 'pain' helps reduce the time spent on development. Such traditional software development methods, which are typically and heavily plan-driven rather than incremental, are usually suitable for large-scale, critical and stable projects where the technology is understood, and requirements can easily be ascertained and are not expected to change or be added to significantly. In contrast, since agile methods involve little or no upfront planning, they are usually characterized by greater uncertainty initially than traditional methods.

However, as Stamelos et al. (2007), pointed out, agile-type developments make it difficult to estimate project costs. Moreover, although agile methods may be suitable for smaller-scale projects, they cannot be scaled easily to larger scale projects, as can traditional methods. It is important to choose the most effective approach to developing software as this can determine the time it will take and costs that are likely to be incurred. When it comes to traceability, Hoang (2015), states that the traceability methods have major drawbacks because they can't properly support the agile methods and still have shortcomings when it comes to waterfall methods, even though they are specifically designed for them. Therefore, it is evident that traceability is affected no matter whether the agile or waterfall method is chosen. SDLC needs to be better understood to address the traceability issue, as some aspects of the aforementioned SDLCs make traceability more difficult. For example, in waterfall, the steps are distinct and may be carried out by different teams allowing ambiguity and confusion to occur

2.3 Requirement Traceability

Traceability can be described as a process used in a software development project by stakeholders to identify the relationship that exists between software artefacts. As mentioned earlier, the association between these artefacts and the way they depend on each other is understood through the process of software development called Requirements Traceability (Requirements.com, 2017). The ability to trace the evolution of an artefact or requirement, in both forward and reverse directions after the description of its links to its life cycle, can be defined as software artefact traceability (Flynt & Salem, 2005). Forward traceability is useful, for instance, when there is a need to investigate the impact of a changed requirement, and backward traceability for understanding a change and investigate the information used to elicit it (Pinheiro & Francisco, 2004).

The process of linking requirements to the issues affecting a project forms the relationship between requirements and traceability. The two are treated separately and are then linked by commonly desired factors (Jarke, 1998). The potential impacts on a user's broken functionality can then be identified through traceability, as the process helps one to identify artefacts, requirements, and links that fall on the matrix of the two (Levy, 2010).

2.4 Issues with Artefacts

Liskin (2015), shows a lack of information about the types of artefacts available, and that there are no widely-agreed types of artefacts defined. It is evident that there is insufficient research on artefact types, so there is a gap in the knowledge of requirement traceability. Therefore, one of the potential contributions of this research is in modelling artefacts and the relationships between them, and to develop a framework which could guide software developers to manage traceability.

3 STUDY DESIGN (METHODOLOGY)

The aim of this paper is to model artefacts and the relationships between them, the people, and SDLC, which would guide software developers to manage traceability. In previous research studies, the problem was that artefacts were mentioned but not linked with the other concepts in the software or discussed in detail. The adopted methodology involves analysis of secondary research to identify types of relationships and the factors that influence them. An ontological analysis using Protégé software, which was chosen for its applicability in managing terminology as well as reasoning Rubin et al. (2007), helped to structure this information on the relationships between the different artefacts using common properties. Such data models ultimately make it easier to understand the elements, relationships, and properties, in this case, the artefacts and the relationships between them, to help develop a new model for requirements traceability

4 RESULTS AND FINDINGS

As indicated previously, data were gathered using over 20 previous studies on artefacts and traceability. For clarity, the results gathered are tabulated (Table 1), which shows the current links and the missing link between requirement traceability artefacts, stakeholders, and SDLC models. The table is designed to show artefact connections, and whether there are any missing and current links.

Table 1: Indications from previous studies

Artefacts	Missing links	Research	Current Links
Requirement to Requirement	✓	[17] [18]	
Requirement Artefacts to Stakeholder		[19]	✓
Requirement to SDLC	✓	[18] [20]	
Requirement Artefacts to each other	✓	[21]	
Requirement to Artefacts		[20]	✓

As shown in table 2, examples of technical artefacts available and commonly encountered include specifications, diagrams, and code. Some types of artefacts include container, generic document, individual element, user-oriented element, solution models, concrete models, abstract models, modelling language, and architectural models. These are different types of artefacts all aimed at assisting in software development and identification of requirements in software development. There are different classifications of artefacts based on architecture, technology-based decisions, modelling language, technical orientation and user-oriented elements (Olga, 2015). The above categorization of the artefacts is the first step in linking them, which forms one of the novel contributions of this paper.

As part of achieving the primary objective of this paper, the basis for the evaluation of the relationships was derived from valid academic research. As an example, Olga (2015), derived a model that categorized requirement artefacts, and although it wasn't very clear and there was some repetition, it did have some ideas that were supportive when building a much more detailed model of the relationships between artefacts. Although handling multiple requirement artefacts can be challenging, the results justified the need to not rely on only a single artefact, and further to ensure the selected artefacts are integrated. The study also showed the possibility of linking multiple different artefacts by constructing a single artefact relating to multiple other artefacts, as when creating specification documents from several elements and models. Moreover, the use of abstract models for mapping the artefacts has considerable potential, such as in identifying dependencies, and helping to make concrete models and thereby improving communication with end users.

Table 2: New categorization of artefact types

Type of Artefacts	Examples
Technical requirements artefacts	specifications, diagrams, code, user-oriented elements, solution models, concrete models, abstract models, modelling language, and architectural models
Business requirement artefacts	Generic document, individual element (user requirement, use cases, user story, and requirement specification

Thus, it can be concluded that contribution of the artefact is an important factor that influences the relationships, but knowledge of the artefacts is a major key player, which requires further investigation in future studies. According to Swathine et al. (2017), "They [the artefacts] are always incomplete and inconsistent due to lack of knowledge".

4.1 Ontology and DL Queries

The ontological structure developed and types of artefacts identified and classified with the help of Olga (2015), previous research, as mentioned in section 4 are discussed in this section. These artefacts include burndown charts, business model diagrams, data models, epic user stories, interaction code, mock-up prototypes, sprint backlogs, task estimations, unit tests, and

user interfaces. The subclasses of two of them are shown as the results of a query. For epic user stories, the subclasses are key requirements, mock-up prototypes, other requirements, and task estimation, and the subclasses of user interfaces are data model, main requirement, other requirement, system requirement, task estimation, and itself. The latter has a direct superclass of requirement artefact. The subclasses for both examples show that the requirement artefacts are interlinked because they contain other artefacts to a greater or lesser degree, and a possible reflexive connection in the case of user interfaces.

Some queries were written to show the results on what the researchers had tested regarding the relationship between requirement artefacts. For example, the query for the main ontology structure was: "Person and manages some Testing_ stage". The results shown were End_ user, Individual, organization, service receiver, stakeholder, and tester_ system_integration. This means there are subclasses and multiple relationships between people who deal with the system and the requirement artefacts.

Another example of the DL query that was done shows the role of each requirement artefact, user stories for example. The query for the Subclasses of the Artefact Epic User Stories is "Requirement_ artefact and MakeUseOf some Epic_User_stories". The subclasses of the query were Main_Requirement, Mock-up_prototype, other_requirement, and Task_estimation, this demonstrated how some requirement artefacts depends on others.

Lastly, as to the direct Subclasses of the Artefact User Interfaces and which ones are interrelated, the query is "isPartOfsomeTesting_Stage". The result is then Data_Model, Main_Requirement, System_ Requirement, Task_estimation, and User_Interfaces. Therefore, all of these needs to be tested in the Testing stage of the SDLC.

4.2 Discussion of Results

In this study, the people who deal with the software were first classified, and the requirement artefacts were identified in detail based on the SDLC. Subsequently, they were structured and then connected. The connections were made explicit based on how they contributed to each other or the system, i.e. on whether or not there were any relationships between them. Examples of these relationships are as follows:

- A People-to-Artefact relationship exists where a stakeholder validates user interfaces whereas developers normally implement them.
- An Artefact-to-Artefact relationship exists where user interfaces rely on requirement specifications whereas the mock-up prototype normally represents user stories.
- An Artefact-to-SDLC relationship exists where the designing enables user interfaces to be developed, whereas requirements gathering normally aids user stories.

The links are naturally transitive; therefore, they could show the effects of issues such as when early errors creep into earlier requirements artefacts through poor handling or potentially through the omission of an important step which prevents information from being effectively linked from one artefact into another. A solution for this would be to come up with a tool which can help people analyze the state of their requirements artefacts, therefore detection of errors can be made and fixed without the software failing.

5 CONCLUSIONS AND FUTURE WORD

In this paper, we proposed a novel design for identifying links between requirement artefacts that could allow software developers to solve traceability problems more efficiently. The motivation for this research was provided by our comparison between the previous studies to find different missing links between requirement artefacts, people dealing with the software, and the SDLC. Previous studies were used due to their academic validity. This paper provides an important foundation towards understanding the relationships between requirement artefacts, people controlling the software, the SDLC, and draws important insights into their future. The main research project works towards building a framework that manages traceability and aids software developers.

REFERENCES

[1] Requirements.com. (2017). An Introduction to Requirements Traceability Requirements.com - Business Requirements, System Requirements, Software Requirements, Requirements Engineering Requirements Wall - Requirements.com. [online] Available at: http://requirements.com/RequirementsWall/tabid/66/articleType/ArticleView/articleId/51/An-Introduction-to-Requirements-Traceability.aspx [Accessed 24 Sep. 2017]

[2] De Lucia, A., Marcus, A., Oliveto, R., & Poshyvanyk, D. (2011). Information Retrieval Methods for Automated Traceability Recovery. *Software and Systems Traceability*, pp.71-98. Doi:10.1007/978-1-4471-2239-5_4.

[3] Levy, Darren. (2010). Why is requirements traceability so important? Gatherspace. Available at http://www.gatherspace.com/static/requirements_traceability.html (accessed February 2017)

[4] Liskin, Olga. (2015). How artefacts support and impede requirements communication. Requirements Engineering Foundation for Software Quality, vol. 9013 of the series *Lecture Notes in Computer Science*, pp. 132-147.

[5] IEEE. (1990). *IEEE standard glossary of software engineering terminology*. Standards Coordinating Committee of the Computer Society of the IEEE. The Institute of Electrical and Electronics Engineers

[6] Young, Ralph R. (2003). *Requirements Engineering Handbook*. Artech House Print.

[7] Miles, C.; M. Hart & M. Gupta. (2012). What does the word artifacts mean in software engineering? – Quora. Retrieved from https://www.quora.com/What-does-the-word-artifacts-mean-in-software-engineering (December 2016).

[8] Liu, Hsiang-Chuan; Wen-Pei Sung& Wenli Yao. (2013). *Information technology and computer application engineering*. Proceedings of the International Conference on Information Technology and Computer Application Engineering. CRC Press.

[9] Wysocki, Robert K. (2011). Effective project management: traditional, agile, extreme. John Wiley & Sons.

[10] Stamelos, Ioannis G. & Sfetsos, Panagiotis. (2007). *Agile software development quality assurance*. Idea Group Inc

[11] Schneider, Jean-Guy & Johnston, Lorraine. (2005). eXtreme Programming—helpful or harmful in educating undergraduates? *Journal of Systems and Software*, vol. 74, issue 2, pp. 121-132

[12] Hoang Duc, V., 2015. Traceability in Agile software projects

[13] Flynt, J. P., & Salem, O. (2005). Software Engineering for Game Developers. Boston, MA: Thomson Course Technology Ptr. – References – Scientific Research Publish. [online] Available at: http://www.scirp.org/(S(351jmbntvnsjt1aadkposzje))/reference/ReferencesPapers.aspx?ReferenceID=1368477 [Accessed 23 Sep. 2017].

[14] Pinheiro, Francisco A. C. (2004). Requirements traceability. Perspectives on Software Requirements, vol. 753 of the series 'The Springer International Series in Engineering and Computer Science', pp. 91-113

[15] Jarke, M. (1998). Requirements tracing. Communications of the ACM, vol. 41, no. 12, pp. 32-36. Doi:10.1145/290133.290145.

[16] Rubin, D.L., Noy, N.F. and Musen, M.A., 2007. Protégé: a tool for managing and using terminology in radiology applications. Journal of digital imaging, 20(1), pp.34-46

[17] Turban, B., Kucera, M., Tsakpinis, A. and Wolff, C., 2009. Bridging the requirements to design traceability gap. In Intelligent Technical Systems (pp. 275-288). Springer Netherlands.

[18] Regan, G., Biro, M., Flood, D. and McCaffery, F., 2015. Assessing traceability—practical experiences and lessons learned. Journal of Software: Evolution and Process, 27(8), pp.591-601.

[19] Sherba, S.A., Anderson, K.M. and Faisal, M., 2003, October. A framework for mapping traceability relationships. In Proceedings of the 2nd International Workshop on Traceability in Emerging Forms of Software Engineering (pp. 32-39).

[20] Hassnain, M. (2015), A Comparative Study on Traceability Approaches in Software Development Life Cycle. ITEE Journal Information Technology & Electrical Engineering. Volume 4, Issue 2.

[21] Swathine, K., Sumathi, N. and Nadu, T. (2017). Study on Requirement Engineering and Traceability Techniques in Software Artefacts. International Journal of Innovative Research in Computer and Communication Engineering, [online] 5(1), p.114. Available at: http://www.ijircce.com [Accessed 19 Mar. 2017].

[22] SDL Forum, & Khendek, F. (2013). *SDL 2013: Model-driven dependability engineering: 16th International SDL Forum, Montreal, Canada, June 26-28, 2013: proceedings*. Berlin: Springer.