Towards Gamification in Software Traceability: Between Test and Code Artifacts

Reza Meimandi Parizi, Asem Kasem and Azween Abdullah
School of Computing and IT, Taylor's University, 47500 Subang Jaya, Selangor, Malaysia
{rezameimandi.parizi, asem.kasem, azween.abdullah}@taylors.edu.my

Keywords: Software Traceability, Software Testing, Validation, Gamification, Reliability.

Abstract: With the ever-increasing dependence of our civil and social infrastructures to the correct functioning of software systems, the need for approaches to engineer reliable and validated software systems grows rapidly. Traceability is the ability to trace the influence of one software artifact on another by linking dependencies. Test-to-code traceability (relationships between test and system code) plays a vital role in the production, verification, reliability and certification of highly software-intensive dependable systems. Prior work on test-to-code traceability in contemporary software engineering environments and tools is not satisfactory and is limited with respect to the need regarding results accuracy, lack of motivation, and high required effort by developers/testers. This paper argues that a new research is necessary to tackle the above weaknesses. Thus, it advocates for the induction of gamification concepts in software traceability, and takes a position that the use of gamificaiton metrics can contribute to software traceability tasks in validating software and critical systems. We propose a research agenda to execute this position by providing a unifying foundation for gamified software traceability that combines self-adaptive, visualization, and predictive features for trace links.

1 INTRODUCTION

Traceability is the ability to describe and follow the life of software artifacts, and is described by the links that connect related artifacts (Lago et al., 2009). Traceability support (for software projects) is deemed to assist software engineers in comprehension, efficient development, and effective management of software systems (Chen and Grundy, 2011). Research has shown that inadequate traceability can be an important contributing factor to software project failures and budget overruns (Dömges and Pohl, 1998); and leads to less maintainable software, and to defects due to inconsistencies or omissions (Winkler and Pilgrim, 2010). On the other hand, achieving affordable traceability can become critical to project success (Watkins and Neal, 1994), and leads to increasing the maintainability and reliability of software systems by making it possible as a means to verify and trace non-reliable parts (Ghazarian, 2010), mostly through testing. Therefore, traceability is important not only for software artifacts, but also as a major component of many standards for software development, such as the CMMI, ISO 9001:2000, and IEEE Std.830-1998, to increase industrial and enterprise considerations.

Coding and testing are two key activities of software development (Huang and Kuo, 2002) that are tightly intermingled, particularly in incremental and iterative methodologies (Rompaey and Demeyer, 2009) such as agile software development. About half of the resources consumed during the software development cycle are testing , verification, and validation resources (Ohtera and Yamada, 1990, Huang and Lyu, 2005, Wang et al., 2010), i.e. testing is expensive and lengthy. It is beneficial to monitor this stage closely to make it more productive, and, if possible, shorter (Kubat and Koch, 1983) in order to develop quality reliable software with affordable resources (Ohtera and Yamada, 1990). In this regard, supplementing testing by traceability can be a decent remedy to provide useful information (e.g. understand the system, help efficiently locate the faulty code, analyze the impact of the changes, find most effective or redundant tests, and rectify defects more reliably in less time) for developers and testers during the testing and debugging phase. Thus, traceability-aware testing, in turn, helps achieve a highly reliable software system by reducing test

effort, and increasing the effectiveness of software fault detection and removal techniques (Kuo et al., 2001, Espinoza and Garbajosa, 2011), especially in critical systems.

However, the trace links between tests (or test cases) and code artifacts (hereafter test-to-code traceability (Parizi et al., 2014)) are typically implicitly presented in the source code, forcing developers towards overhead code inspection or name matching to identify test cases relevant to a working set of code components. Likewise, tests also require frequent adaptation to reflect changes in the production code to keep an effective regression suite (Rompaey and Demeyer, 2009), e.g. during software evolution. Identifying trace links can be seen as an arduous, time-consuming, high cost, and error-prone job (Sundaram et al., 2010, Mader et al., 2013). Thus, capturing and creating traceability information as a by-product of development (i.e. performing in parallel with the artifacts / proactive) is often found (Sundaram et al., 2010) to be tedious to developers/testers, and is rarely done to the necessary level of granularity, which can result in poor quality and incomplete recording of the relevant traceability.

Existing traceability approaches (Antoniol et al., 2002, Marcus and Maletic, 2003, Egyed, 2003, Wang et al., 2009, Lucia et al., 2007, Witte et al., 2008, Jirapanthong and Zisman, 2009, Bacchelli et al., 2010) mostly rely on post-mortem analysis (ad hoc and after the fact) of software artifacts as a sought-after activity to recover trace links. This is also the case with existing test-to-code traceability approaches (see Section 2), where traceability is not an internal characteristic or property of the source code or tests. Instead, traceability is achieved outside the source code and test artifacts, after they are built. It has to be said however that obtaining traceability information using this strategy suffers from a technical bias that impedes its performance (including accuracy of results and required effort and cost) in modern software projects, e.g. the emerging trend towards the adoption of agile-based or big data-driven projects which are the mainstream.

Much of test-to-code traceability problems in this case, first and foremost, derive from the fact that for most software projects, ensuring traceability is a duty honour'd in the breach and is not regarded as a visible and feasible option to individual stakeholders or the project manager (Mader et al., 2013). Also, it can be contended that the current challenges inherent to test-to-code traceability originate in a lack of motivation and improper engagement of human developers/testers in traceability tasks.

To date, the realization of test-to-code traceability has received lesser attention in which only a small proportion of research work in the literature targets this subject. Even in industrial settings, test-to-code traceability is not common in software development (Qusef et al., 2014), and often falls far short of accepted principles in software engineering. These observations show that the current research and practice on test-to-code traceability is in infancy. Hence, to support the advancement in this area, and the maturity of the test-to-code traceability, new approaches need to be developed or current approaches evolved with new concepts and ideas. This in turn provided a motivation for this research to propose a change of research focus from existing post-mortem recovery of trace links to proactive construction of traceable software systems with highly engaged human factors.

Gamification (Deterding et al., 2011) as a new technological trend, introduces game mechanisms into non-gaming contexts in order to increase engagements, motivation, and participation. Gamification has recently been a burgeoning interest for researchers and enterprises due to its merits (Koivisto and Hamari, 2014). An examination of the literature reveals that gamification has been utilized across various fields of research and domains, aiming for enhancement of conventional techniques. For instance, in requirement elicitation (Fernandes et al., 2012), software architecture (Herzig et al., 2012), testing environment (Chen, 2011), learning and education (Domínguez et al., 2013, Simões et al., 2013, de-Marcos et al., 2014, Rodríguez Corral et al., 2014), online business and retail (Insley and Nunan, 2014), tablet software (Browne et al., 2014), and government services (Bista et al., 2014). The results achieved from these studies and some SE-specific work (Dubois and Tamburrelli, 2013, Vasilescu, 2014, Pedreira et al., 2015) motivated the use of gamification and its usefulness within software and systems engineering activities. In this respect, the idea behind gamification (as it creates feedback-rich environment) can be worthwhile, attractive and offers much promise regarding test-to-code traceability human-related problems.

Despite the large number of recent studies on the utilization of different gamification concepts, there has been no sound and fundamental work for which this concept is well suited to test-to-code traceability of software-intensive systems. Thus, investigating this idea to the traceability problem remains

challenging as an open research area to be conducted. This research will address this gap in the existing traceability literature. As far as we are aware, this is the first attempt made in the literature to fundamentally bridge software traceability field with gamified concepts.

2 LITERATURE REVIEW

In this section, we review the existing approaches available for test-to-code traceability and assess their strengths and weaknesses. A general standpoint to review current approaches is to look into underlying strategies to achieve traceability, which are generally categorized in two ways (Sundaram et al., 2010):

1. By-product of artifact development (a.k.a proactive)
2. After-the-fact artifact development

Developers/testers are not usually willing to record traceability information while they are developing the software systems. If traceability information is maintained as a by-product of development throughout a project's lifetime, it would be desirable and highly recommended; but unfortunately researchers have not given much attention to this category (Sundaram et al., 2010, Mader et al., 2013). As a result, traceability is often conducted in an ad-hoc, after-the-fact manner and, therefore, its benefits are not always fully realized (Cleland-Huang et al., 2014). Researchers in (Deursen and Moonen, 2002, Sneed, 2004, Rompaey and Demeyer, 2009, Qusef et al., 2010, Cleland-Huang et al., 2014) argued that in most cases of real-world software systems, traceability links are not maintained properly or even documented, and thus are unavailable during the maintenance phase; therefore, as a last resort, traceability links need to be recovered during the evolution of software systems.

In the last decade, enormous approaches have been proposed to recover traceability links between various software artifacts. But, the realization of test-to-code traceability has received lesser attention in which only a small proportion of research work in the literature targets this subject. In the following, the existing test-to-code traceability recovery approaches are reviewed and discussed.

Today's integrated development environments provide little support for software developers to navigate between unit tests and tested classes. As an example, Microsoft visual studio 2010 environment provides a wizard to create unit tests for existing classes. In its subsequent versions, a unit test generator is available as an extension for users. The Eclipse Java environment also provides the same feature. Several sources (Sneed, 2004, Rompaey and Demeyer, 2009, Qusef et al., 2010, Espinoza and Garbajosa, 2011, Qusef et al., 2011, Qusef et al., 2014) have proposed different test-to-code traceability solutions to recover traceability links between unit tests and production code.

Rompaey and Demeyer, in 2009, empirically compared a series of automatable traceability approaches, namely Naming Convention (NC), Fixture Element Types (FET), Static Call Graph (SCG), Last Call Before Assert (LCBA), Lexical Analysis (LA) and Co-Evolution. The results of their empirical experiments on three open-source Java systems, JPacman, ArgoUML, and Mondrian, indicated that although LCBA, LA and FET have a high applicability, they have poor results in precision and recall. On the other side, NC showed a good result in accuracy. The best overall result was achieved with a combination of NC, FET and LCBA.

Qusef et al., in 2010, introduced a traceability recovery approach based on Data Flow Analysis (DFA) to enhance LCBA method proposed by (Rompaey and Demeyer, 2009). This approach works based on the assumption that, if a method call in a unit test affects the result of the last assert statement, the method should belong to the unit under test. To assess the accuracy of DFA, Qusef et al performed a case study on AgilePlanner and Mondrian which are two open-source Java software systems. In this case study, the effectiveness of DFA in identification of test-to-code traceability links were analyzed and DFA was compared with NC and LCBA, in terms of accuracy of the recovered traceability links. The results showed that DFA is more accurate than NC and LCBA. However, the proposed approach failed when there was no real relationship between unit test classes and classes under test that affect the results of the last assert assessment.

Qusef et al. also presented (Qusef et al., 2011, A.Qusef et al., 2012, Qusef et al., 2014) another approach called SCOTCH (Slicing and Coupling based Test to Code trace Hunter) which is based on the last assert statement analysis, dynamic slicing and conceptual coupling together to identify a set of tested classes in two sequential steps. In the first step, SCOTCH identifies the last assert assessment instance in each execution trace using dynamic slicing. In the second step, it uses a stop-class list (list of classes from standard library types such as

java.*, javax.*, org.junit.*) and conceptual coupling to shortlist the candidate types and prune-out unwanted classes. Since only conceptual coupling is used as a filtering strategy and there are more crucial information in the classes, most recently, Qusef et al. ,in 2014, introduced an extension to this approach, called SCOTCH+. This extended approach employs a more refined filtering strategy based on both internal and external textual information, and shows some improvement compared to the original version.

The review of the existing test-to-code traceability recovery approaches provided us with a number of lessons to highlight the existing problems in this field. The major problems/issues that were identified are as below:

- Test-to-code traceability links are not established completely in all existing traceability recovery approaches (missing and redundant links).
- Most approaches recover traceability links between unit tests and the classes under test, and not specifically other constructs such as methods (capturing only high-level/coarse-grained trace links).
- All the recovered test-to-code traceability links need to be verified manually by a domain expert (high cost).
- There is no delivery of visual analytics and support for trace links (lack of visualization support).
- The Last, and most importantly, is non-interactiveness in the sense that the approaches are not self-adaptive as the project evolves, thus requiring high manual effort to maintain trace links (high effort).

As a summary, it has to be said that there is a lack of ubiquitous test-to-code traceability foundation and approach to recover all traceability links between tests and production code of software systems with minimal human effort for links verification. As the analysis of the code is a sophisticated task, in this research we propose a strategic and engaging approach built into the project environment for achieving traceability information with minimal effort for developers to facilitate software testing/maintenance. In contrast to our proposed research, none of the previous related work has given attention to by-product way of achieving traceability to make use of the rich human resource (developers/testers) on the spot.

3 RESEARCH AGENDA

3.1 Research Objectives

The main objective of this research is to develop a test-to-code traceability system for software-intensive applications by building a proactive and self-adaptive model inspired by gamification mechanics, and one that is capable of creating and maintaining trace links throughout the project lifetime. The specific objectives are as follows:

1. To define a strategic theory that identifies the role of gamification, the use of game mechanics, in enabling developer engagement with traceability tasks.
2. To derive and design a ubiquitous formal model based on the theory in (1) that self-adapts itself as a project evolves and also learns from human feedback with predictive and visualization features.

3.2 Research Process

The research design (planning) below provides the overall structure of the procedures and steps that need to be done in order to achieve the two stated objectives of this research.

1) Objective 1: As the increasing application of social and mobile games plays important trends in culture today, we were originally inspired to understand the value of adding the elements of games into traceability tasks. For a software-intensive system to start and proceed on to a sustainable operation, it is important that developers are encouraged to contribute positively and frequently in traceability tasks. To achieve this objective, we consolidate the different gamification design elements and demographics to explore traceability task of developers. In this regard, we review the literature about game design and identify motivational elements that have been incorporated in the design of gamification. Then, using case analysis and collecting, analyzing, and processing every piece of evidence from which trace data can be inferred and managed, we verify these elements and will propose a game-thinking theory for traceability purposes. We use the set theory and z-notation to specify the roles, functions and responsibilities of each involving agent (i.e. developer/tester).

Furthermore, for the formalization and representation of the traceability concepts defined in the proposed gamified theory and the project's artifacts (i.e. test and code), we use the F-logic

(frame logic) language (Kifer and Lausen, 1989). We choose F-logic because it is a knowledge representation and ontology language, which combines the advantages of conceptual modeling with object-oriented, frame-based languages and offers a declarative and compact syntax. Features include object identity, complex objects, inheritance, polymorphism, query methods, and encapsulation. F-logic allows us to define the required complex data schema, to reason about the differences between two schemas, to reason about differences between a schema and instances, and to infer implicit knowledge from a schema.

2) Objective 2: To achieve this objective, we devise the following components, whose inner workings are discussed.

a) *Self-adaption*: Self-aware systems are able to modify their own behavior in an attempt to optimize performance. Such systems can self-diagnose, self-repair, adapt, add or remove software components dynamically. To work toward achieving automated adaptation in trace creation and maintenance, we undertake the following activities:

• Develop intelligent tracing solutions which are not constrained by the terms in the source and target artifacts, but which understand game-specific concepts (proposed in the theory), and can reason intelligently about relationships between artifacts.

• Adopt self-adapting solutions, which are aware of the current project state and reconfigure accordingly in order to optimize the quality of trace links. To discover the best way to model available features of the trace engine (e.g. stoppers, stemmers, VSM, LSI, LDA, voters, etc.) in a feature model, we use a genetic algorithm to search for the ideal configuration.

b) *Human Feedback*: Analysts are needed to evaluate the generated trace. Human feedback can impact the quality of the generated trace links. Because in many domains, especially safety- and mission-critical ones, results from automated techniques cannot be used unless inspected by a human. We therefore need to gain a better understanding of the process human analysts go through to vet and approve a generated trace link, and to understand which sequences of actions are more or less likely to lead to correct decisions. Furthermore, with the increasing adoption of social collaboration tools, it is interesting to explore the impact of collaborative decision making on the correctness of trace links.

As a novel initiation, in this research, we intend to integrate feedback provided by gamification data to be useful to enrich the accuracy of results. To achieve this, we adopt to use a standard information technique known as Rocchio, which increases or decreases term weightings used to compute similarity scores according to whether a term occurs in a rejected or confirmed trace link. In this case, developers may also directly modify the trace query by adding and/or removing terms and the 'before' and 'after' queries can be used to learn a set of query transformation rules which can be used to improve future queries.

c) *Predictive and monitoring feature*: To address this feature, we use aspect-orientation methodology and time series analysis to the runtime monitoring and quality forecasting of trace links, as a component. Statistical techniques for analyzing time series will be used to facilitate the prediction and forecasting (the term 'prediction' and 'forecasting' are interchangeably used) of probabilistic quality link properties, most importantly trace coverage metrics. Furthermore, in order to reduce the human effort and to cope with more sophisticated scenarios, in this step we aim to automate the analysis and modeling process by utilizing relevant tools such as Matlab, AMOS, SPSS. This part will deliver a prospective trace capture solution that will be capable of monitoring development environments, including artifacts and human activities, to infer trace-related information.

d) *Visualization feature*: To address this feature, we generate trace slice visualizations in which the test case is the root node and all direct and indirect traced artifacts that contribute to mitigating the associated bug will be shown as a tree. More specifically, linear, tabular, hierarchical, and graph-based representations, as the most prevalent forms of visualization component, will be used to depict traceability information in the proposed approach, and hyper-links (cross-references) are routinely used to associate artifacts and traverse the links interactively.

In summary, it is worth highlighting the distinguishing characteristics of the proposed work over the existing work, which are in line with the future vision of software traceability. These include being engaging (brought by gamification), by-product, self-adaptive, trace visual-equipped, and trace monitor-equipped.

3.3 Empirical Studies

To assess the model in a context of test-to-code traceability, the research: (1) Adopts a qualitative methodology carrying out in-depth interviews with

individuals who are experienced developers/testers. Further study is planned to evaluate the acceptance of the proposed approach by collecting responses from participants regarding their perceptions of the elements of the gamification activities. It is expected that results from this evaluation will provide a basis for software enterprises to leverage the proposed theory and approach properly. (2) Provides a large-scale experiment to statistically compare the current test-to-code traceability approaches, using measurements to determine the effectiveness and efficiency of each approach. In this case, effectiveness refers to the accuracy of each approach in constructing links and efficiency refers to the effort to produce links by each approach. This evaluation includes:

1) *Define variables and measures:* In this step, we define the independent and dependent variables as well as, metrics to be used to quantify the dependent variables. For instance, to evaluate and compare the accuracy of each approach, we use two well-known, recall and precision metrics as well as, F-measure to assess the global accuracy of the approaches.

2) *Determine subject programs or datasets:* As with any experimental setting in software engineering, here we also require a set of subject programs and/or dataset to collect data through observation of its execution process. We use a suite of four large-scale software system projects (named AspectJ, Rhino, ArgoUML, BlueJ) written in Java and taken from considerably different application domains as objects in the experiment. We required all four systems to share the following properties:

- Open source — To enable replications and secondary studies of our experiment. Furthermore, to provide access to the source code as to verify the presence of test suite and to apply the given automated approaches.
- Written in Java — To establish consistency among the system chosen for the experiment. In addition, to comply with implementations related to the existing approaches as they are targeted towards systems developed in Java.
- Possess test suite — To enable application of the approaches and aggregate measurements.

3) *Sketch of experimental design:* We use a randomized block design in the experiment. We block the experiment on the subjects in which each one is used exactly once to measure the effect of every treatment allocated in a random order. By using this design, the variability will be divided into variability due to treatments and variability due to blocks. Thus, the effect of the treatments could be investigated without interfering from the effect of blocks masking the outcome of the experiment.

After the experimental operation, proper and robust statistical tests will be used to draw the conclusions with respect to the effectiveness and the efficiency of the proposed approach over peer approaches.

3.4 Implementation

As a proof-of-concept development, the proposed approach as a gamification plugin will be prototypically incorporated into well-known frameworks such as .NET and Java IDEs (e.g. Eclipse). Such plugin will contain a dashboard that displays the tracing progress for a project for tracking and managing the project's tracing goals and also for motivating team members to create appropriate trace links. The dashboard will display useful information such as burn down charts showing the percentage of hazards that do not have mitigating requirements, or the percentage of mitigating requirements without passing test cases. This information will be generated via trace queries. Personalized views will be created for individual project members.

4 CONCLUDING REMARKS

Traceability is an increasingly common element of public and private systems for monitoring compliance with excellence, as quality of deployed system's source code is ultimately of utmost importance. The current challenges inherent to test-to-code traceability originate in a lack of motivation and improper engagement of human developers/testers in traceability tasks.

This position paper has presented a research agenda in order to analyze and improve test-to-code traceabilty from a human based-perspective. The agenda is mainly based on gamification concept, which aim to build a foundation that combines self-adaptive features. The expected outcome of this research is a unifying theory for gamified software traceability that can be used as a template to enhance software development processes and languages. The completion of this theory would also create opportunities to explore new ways of detecting traceability coverage/failure in safety-critical software systems such as aeronautics, medical

devices, and railway communications to demonstrate if a rigorous process has been followed and to validate that the system is safe for its intended use.

ACKNOWLEDGEMENTS

The research leading to this paper (based on a project proposal) has received Fundamental Research Grant Scheme funding from the Ministry of Education, Malaysia, with reference FRGS/2/2014/ICT01/TAYLOR/03/1.

REFERENCES

A.Qusef, Bavota, G., Oliveto, R., Lucia, A. D. & Binkley, D. 2012. Evaluating Test-To-Code Traceability Recovery Methods Through Controlled Experiments. *Journal Of Software: Evolution And Process,* 25**,** 1167–1191.

Antoniol, G., Canfora, G., Casazza, G., Lucia, A. D. & Merlo, E. 2002. Recovering Traceability Links Between Code And Documentations. *IEEE Transactions On Software Engineering,* 28**,** 970-983.

Bacchelli, A., Lanza, M. & Robbes, R. Linking E-Mails And Source Code Artifacts. Proceedings Of The 32nd Acm/Ieee International Conference On Software Engineering, 2010 Cape Town, South Africa. 1806855: Acm, 375-384.

Bista, S. K., Nepal, S., Paris, C. & Colineau, N. 2014. Gamification For Online Communities: A Case Study For Delivering Government Services. *International Journal Of Cooperative Information Systems* 23.

Browne, K., Anand, C. & Gosse, E. 2014. Gamification and Serious Game Approaches For Adult Literacy Tablet Software. *Entertainment Computing,* 5**,** 135-146.

Chen, N. Gate: Game-Based Testing Environment. Proceedings Of The 33rd International Conference On Software Engineering, 2011. 1078-1081.

Chen, X. & Grundy, J. Improving Automated Documentation To Code Traceability By Combining Retrieval Techniques. Proceedings Of The 26th Ieee/Acm International Conference On Automated Software Engineering, 2011. 2190177: Ieee Computer Society, 223-232.

Cleland-Huang, J., Gotel, O. C. Z., Hayes, J. H., Mader, P. & Zisman, A. Software Traceability: Trends And Future Directions. Proceedings Of The 36th International Conference On Software Engineering, 2014. Acm, 55-69.

De-Marcos, L., Domínguez, A., Saenz-De-Navarrete, J. & Pagés, C. 2014. An Empirical Study Comparing Gamification And Social Networking On E-Learning. *Computers & Education,* 75**,** 82-91.

Deterding, S., Dixon, D., Khaled, R. & Nacke, L. from Game Design Elements To Gamefulness: Defining "Gamification". Proceedings Of The 15th International Academic Mindtrek Conference: Envisioning Future Media Environments, 2011 Tampere, Finland. Acm, 9-15.

Deursen, A. V. & Moonen, L. The Video Store Revisited– Thoughts On Refactoring And Testing. Proceedings Of 3rd International Conferance, Extreme Programming And Flexible Processes In Software Engineering, 2002. 71-76.

Dömges, R. & Pohl, K. 1998. Adapting Traceability Environments To Project-Specific Needs. *Communications Of The Acm,* 41**,** 54-62.

Domínguez, A., Saenz-De-Navarrete, J., De-Marcos, L., Fernández-Sanz, L., Pagés, C. & Martínez-Herráiz, J.-J. 2013. Gamifying Learning Experiences: Practical Implications And Outcomes. *Computers & Education,* 63**,** 380-392.

Dubois, D. J. & Tamburrelli, G. Understanding Gamification Mechanisms For Software Development. Proceedings Of The 9th Joint Meeting On Foundations Of Software Engineering, 2013 Saint Petersburg, Russia. 2494589: Acm, 659-662.

Egyed, A. 2003. A Scenario-Driven Approach To Trace Dependency Analysis. *Ieee Transactions On Software Engineering,* 29**,** 116-132.

Espinoza, A. & Garbajosa, J. 2011. A Study To Support Agile Methods More Effectively Through Traceability. *Innovations In Systems And Software Engineering,* 7**,** 53-69.

Fernandes, J. A., Duarte, D., Ribeiro, C., Farinha, C., Pereira, J. A. M. & Silva, M. M. D. 2012. Ithink : A Game-Based Approach Towards Improving Collaboration And Participation In Requirement Elicitation. *Procedia Computer Science* 15 66 -77.

Ghazarian, A. 2010. A Research Agenda For Software Reliability. *Ieee Transactions On Reliability, Ieee Reliability Society Technical Operations Annual Technical Report For 2010,* 59**,** 449-482.

Herzig, P., Ameling, M. & Schill, A. A Generic Platform For Enterprise Gamification. Proceedings Of The Joint Working Ieee/Ifip Conference On Software Architecture And 6th European Conference On Software Architecture, 2012. 219-223.

Huang, C.-Y. & Kuo, S.-Y. 2002. Analysis Of Incorporating Logistic Testing-Effort Function Into Software Reliability Modeling. *Ieee Transactions On Reliability,* 51**,** 261-270.

Huang, C.-Y. & Lyu, M. R. 2005. Optimal Testing Resource Allocation, And Sensitivity Analysis In Software Development. *Ieee Transactions On Reliability,* 54**,** 592-603.

Insley, V. & Nunan, D. 2014. Gamification And The Online Retail Experience. *International Journal Of Retail & Distribution Management,* 42**,** 340-351.

Jirapanthong, W. & Zisman, A. 2009. Xtraque: Traceability For Product Line Systems. *Software And System Modeling,* 8**,** 1619-1366.

Kifer, M. & Lausen, G. F-Logic: A Higher-Order Language For Reasoning About Objects, Inheritance, And Scheme. Proceedings Of The Acm Sigmod

International Conference On Management Of Data, 1989. 134-146.

Koivisto, J. & Hamari, J. 2014. Demographic Differences In Perceived Benefits From Gamification. *Computers In Human Behavior,* 35**,** 179-188.

Kubat, P. & Koch, H. S. 1983. Managing Test-Procedures To Achieve Reliable Software. *Ieee Transactions On Reliability,* R-32**,** 299-303.

Kuo, S.-Y., Huang, C.-Y. & Lyu, M. R. 2001. Framework For Modeling Software Reliability, Using Various Testing-Efforts And Fault-Detection Rates. *Ieee Transactions On Reliability,* 50**,** 310-320.

Lago, P., Muccini, H. & Vliet, H. V. 2009. A Scoped Approach To Traceability Management. *Journal Of Systems And Software,* 82**,** 168–182.

Lucia, A. D., Fasano, F., Oliveto, R. & Tortora, G. 2007. Recovering Traceability Links In Software Artifact Management Systems Using Information Retrieval Methods. *Acm Transactions On Software Engineering And Methodology,* 16**,** 13:1-13:50.

Mader, P., Jones, P. L., Zhang, Y. & Cleland-Huang, J. 2013. Strategic Traceability For Safety-Critical Projects. *Ieee Software,* 30**,** 58-66.

Marcus, A. & Maletic, J. I. Recovering Documentation-To-Source-Code Traceability Links Using Latent Semantic Indexing. Proceedings Of The 25th International Conference On Software Engineering, 2003 Portland, Oregon. Ieee Computer Society, 125-135.

Ohtera, H. & Yamada, S. 1990. Optimal Allocation & Control Problems For Software-Testing Resources. *Ieee Transactions On Reliability,* 39**,** 171-176.

Parizi, R. M., Lee, S. P. & Dabbagh, M. 2014. Achievements And Challenges In State-Of-The-Art Software Traceability Between Test And Code Artifacts. *Ieee Transactions On Reliability,* 63**,** 913-926.

Pedreira, O., García, F., Brisaboa, N. & Piattini, M. 2015. Gamification In Software Engineering – A Systematic Mapping. *Information And Software Technology,* 57**,** 157–168.

Qusef, A., Bavota, G., Oliveto, R., De Lucia, A. & Binkley, D. Scotch: Test-To-Code Traceability Using Slicing And Conceptual Coupling. Proceedings Of The 27th Ieee International Conference On Software Maintenance 25-30 Sept. 2011 2011. 63-72.

Qusef, A., Bavota, G., Oliveto, R., Lucia, A. D. & Binkley, D. 2014. Recovering Test-To-Code Traceability Using Slicing And Textual Analysis. *Journal Of Systems And Software,* 88**,** 147-168.

Qusef, A., Oliveto, R. & De Lucia, A. Recovering Traceability Links Between Unit Tests And Classes Under Test: An Improved Method. Ieee International Conference On Software Maintenance (Icsm), 2010. 1-10.

Rodríguez Corral, J. M., Civit Balcells, A., Morgado Estévez, A., Jiménez Moreno, G. & Ferreiro Ramos, M. J. 2014. A Game-Based Approach To The Teaching Of Object-Oriented Programming Languages. *Computers & Education,* 73**,** 83-92.

Rompaey, B. V. & Demeyer, S. Establishing Traceability Links Between Unit Test Cases And Units Under Test. Proceedings Of The 2009 European Conference On Software Maintenance And Reengineering, 2009. 1545440: Ieee Computer Society, 209-218.

Simões, J., Redondo, R. D. & Vilas, A. F. 2013. A Social Gamification Framework For A K-6 Learning Platform. *Computers In Human Behavior,* 29**,** 345-353.

Sneed, H. M. Reverse Engineering Of Test Cases For Selective Regression Testing. Proceedings Of The 8th European Conference On Software Maintenance And Reengineering, 2004. 69-74.

Sundaram, S. K., Hayes, J. H., Dekhtyar, A. & Holbrook, E. A. 2010. Assessing Traceability Of Software Engineering Artifacts. *Requirements Engineering,* 15**,** 313-335.

Vasilescu, B. Human Aspects, Gamification, And Social Media In Collaborative Software Engineering. Proceedings Of The 36th International Conference On Software Engineering, 2014 Hyderabad, India. 2591091: Acm, 646-649.

Wang, X., Lai, G. & Liu, C. 2009. Recovering Relationships Between Documentation And Source Code Based On The Characteristics Of Software Engineering. *Electronic Notes In Theoretical Computer Science,* 243**,** 121-137.

Wang, Z., Tang, K. & Yao, X. 2010. Multi-Objective Approaches To Optimal Testing Resource Allocation In Modular Software Systems. *Ieee Transactions On Reliability,* 59**,** 563-575.

Watkins, R. & Neal, M. 1994. Why And How Of Requirements Tracing. *Ieee Software,* 11**,** 104-106.

Winkler, S. & Pilgrim, J. V. 2010. A Survey Of Traceability In Requirements Engineering And Model-Driven Development. *Software And Systems Modeling,* 9**,** 529-565.

Witte, R., Li, Q., Informatic, F. F., Zhang, Y. & Rilling, J. 2008. Text Mining And Software Engineering: An Integrated Source Code And Document Analysis Approach. *Iet Software* 2**,** 1-19.