On the gamification of human-centric traceability tasks in software testing and coding

Conference Paper · June 2016

DOI: 10.1109/SERA.2016.7516146

1 author:

Reza M. Parizi
Kennesaw State University, Atlanta, USA
173 PUBLICATIONS **3,867** CITATIONS
SEE PROFILE

On the Gamification of Human-Centric Traceability Tasks in Software Testing and Coding

Reza Meimandi Parizi
School of Engineering and Computing Sciences
New York Institute of Technology
Nanjing Campus, China
rparizi@nyit.edu

Abstract—**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 dependable systems. In practical settings, however, traceability tasks are most often observed in the breach by human developers/testers. Prior research works on test-to-code traceability in software engineering do not provide high traceability output and accuracy as they mainly rely on sought-after approaches to recover links. Gamified Software Engineering (GSE) is a growing field that in particular taps into gamification, the application of game mechanics in non-game contexts, to address human-related concerns in the field of SE. This paper argues that a new gamified approach is necessary to tackle the human issue of capturing traceability information in a by-product manner. Thus, it advocates for the induction of gamification concepts in software traceability. We propose a conceptual framework where the gamification is infusing engagement into human-centric traceability tasks to record trace links. An empirical evaluation was performed to assess the performance of the framework compared with a state-of-the-art approach.**

Keywords—Testing; Traceability; Gamification; Code; Reliability; Motivation

I. Introduction

Traceability is the ability to describe and follow the life of software artifacts, and is described by the links that connect related artifacts [1]. Traceability support (for software projects) is deemed to assist software engineers in comprehension, efficient development, and effective management of software systems [2]. Research has shown that inadequate traceability can be an important contributing factor to software project failures and budget overruns [3]; and leads to less maintainable software, and to defects due to inconsistencies or omissions [4]. On the other hand, achieving affordable traceability can become critical to project success [5], 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 [6], 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 [7] that are tightly intermingled, particularly in incremental and iterative methodologies [8] such as agile software development. About half of the resources consumed during the software development cycle are testing, verification, and validation resources [9-11], i.e. testing is expensive and lengthy. It is beneficial to monitor this stage closely to make it more productive, and, if possible, shorter [12] in order to develop quality reliable software with affordable resources [9]. 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 [13, 14], especially in critical systems.

However, the trace links between tests (or test cases) and code artifacts (hereafter test-to-code traceability [15]) 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 [8], e.g. during software evolution. Identifying trace links can be seen as an arduous, time-consuming, high cost, and error-prone job [16, 17]. Thus, capturing and creating traceability information as a by-product of development (i.e. performing in parallel with the artifacts / proactive) is often found [16] 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 [18-25] 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 II), 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 that 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 honored in the breach and is not regarded as a visible and feasible option to individual stakeholders or the project manager [17]. In addition, 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 [26], 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 [27] 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 [28]. 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 [29], software architecture [30], testing environment [31], learning and education [32-35], online business and retail [36], tablet software [37], and government services [38]. The results achieved from these studies and some SE-specific work [39-41] 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. Inspired by this, this paper hence investigates the application of gamification to the traceability problems by proposing a conceptual framework that aims to engage developers in related tasks for more quality tracing results.

The rest of this paper is structured as follows: Section II presents related work and background. Section III gives the details of the framework, including its functional and conceptual features. Section IV briefly describes the implementation part of the framework. Section V discusses the empirical assessment and results. Section VI gives the conclusion and future work.

II. Related work and background

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 [16, 17]. As a result, traceability is often conducted in an ad-hoc, after-the-fact manner and, therefore, its benefits are not always fully realized [42]. Researchers in [8, 42-46] 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.

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 [8, 14, 26, 44, 45, 47] have proposed different test-to-code traceability solutions to recover traceability links between unit tests and production code.

Rompaey and Demeyer [8], 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. [45], introduced a traceability recovery approach based on Data Flow Analysis (DFA) to enhance LCBA method proposed by [8]. 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, the authors 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 was 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 [26, 47, 48] 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. [26], 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 follows:

• 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 tackle quality 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 an 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.

III. Gamified Traceability Framework

Gamification [41] seeks for improvement of the user's engagement, motivation, and performance when carrying out a certain task, by means of incorporating game mechanics and elements, thus making that task more productive. In fact, we see the widespread use of gamification in the real world today in the realms of learning, business and even the routine aspects of lifestyle. For instance, the privileges of online shopping with credit cards, social media (e.g. Linkedin[1] progress indicator or endorsement buttons), e-learning environments (e.g. Code Academy that uses a motivational system of various game mechanics to keep learners engaged and incentivized to continue learning), mobile apps, and many more. Hence, gamification potentially makes technology more inviting by virtue of users' engagement to desired behaviors.

Software engineering (SE) is inherently a human-centric and collaborative process in a quite volatile environment. This nature in turn makes the gamification a good fit to improve the daily engagement and motivation of software engineers in their tasks. Seen it this way, gamification can be realized as a remedy to unlock motivations of the currently unengaged software engineering-related concerns/ problems.

As the focus of this paper concerns, a SE-related problem in software traceability between code an test artifacts is used to testify the application of gamification. For a software-intensive system to start and proceed on to a sustainable and reliable operation, it is important that developers are encouraged to contribute positively and frequently in traceability tasks. However, as mentioned earlier, capturing and creating traceability data as a by-product of development (i.e. performing in parallel with the artefacts) and maintaining the links over time is like a de-motivator for people involved, and is rarely done to the necessary level of granularity by software engineers. Given this, now a question arises: Does this problem have anything to do with the human software engineers involved? Yes, stemmed from the lack of motivation and engagement! Thus, it could be streamlined by incorporating gamification in traceability tasks.

The main contribution in this research is the inclusion of the game concepts to traceability tasks to show how the concept of gamification can be used for more quality results. As the analysis of the code is a sophisticated task, in this research we propose a by-product and engaging framework built into the project environment for achieving traceability information with minimal effort for developers to facilitate software testing/maintenance.

A. *Gamification Design*

There are various aspects to consider and support in the gamification of a software problem (a.k.a. gamification design), including: understanding the problem and its best fit to gamification, knowing the target audience of the gamified environment, tailoring the context to fulfill different types of players' satisfaction, and most importantly having meaningful *game mechanics* to put into use the gamified concept.

Defining game mechanics (i.e. suitable dynamics for taking advantage to engage and motivate players in the best way) is the most challenging part of a gamified model as we seek to turn gamification into a key asset. To derive the suitable game mechanics (presented in Table I), we consolidated the different gamification design elements and demographics to explore traceability task of developers. In this regard, we reviewed the literature about game design and identified motivational elements that have been incorporated in the design of gamification in various studies, particularly those related to software development environment.

B. *The Framework*

In this framework, we conjecture that while tests are developed, the correct traceability information will be

[1] https://www.linkedin.com/

incorporated or maintained by test developers (driven by the power of game mechanics as the purpose of the game). The rationale behind this intuitive idea was inspired by the fact that test developers are well aware of the method or class for which they are writing a test. Thus, they can optimally embed traceability links at the same time developing the test projects. As such, if traceability information is embedded in the software development phase, no post-verification of recovered links is needed in later phases.

The proposed framework takes part in two main phases of software life cycle: development phase (when tests are developed and traceability information embedded) and maintenance phase (when the system is up and running) as shown in Fig.1.

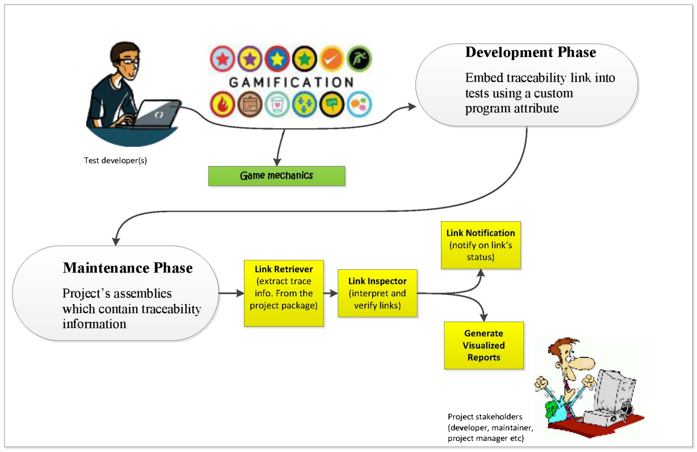

GAMIFICATION
Development Phase
Embed traceability link into tests using a custom program attribute
Test developer(s)
Game mechanics
Maintenance Phase
Project's assemblies which contain traceability information
Link Retriever (extract trace info. From the project package)
Link Inspector (interpret and verify links)
Link Notification (notify on link's status)
Generate Visualized Reports
Project stakeholders (developer, maintainer, project manager etc)

Fig. 1. The cenptual view of the framework

As a programmatic means to help embed traceability links directly and explicitly into the test code, a custom attribute, called traceability attribute, is introduced which acts as a direct and explicit traceability link (this is the procedure for action as the method of play). This traceability attribute (as a snippet of code) is incorporated into every test case through the test development process in order to record traceability information. The traceability attribute can be incorporated easily by test developers at the same time of writing a test case. Therefore, incorporating the attributes is performed at the test development phase, as a by-product of development.

TABLE I. LIST OF GAME MECHANICS

Mechanics	**Effects/ benefits**
Points	Extrinsic rewards and achievement stats
Feedback	Connecting and communal collaboration and discovery
Reputation building	Peer-pressure and ranks based on level of skill associated. The competition is an element of it
Avatar	Protect privacy issues
Time spent	Progress and tracking
Quests/Challenges	Challenges usually imply a time limit to complete the task, whereas Quests are meant to be a journey of obstacles a tester must overcome

To provoke feelings of motivation and engagement in developers, there are six types of game mechanics at different levels of abstractions utilized in the framework (see Table I). The framework requires making use of these game mechanics using a kind of vector model, where each dimension corresponds to a separate mechanic.

In the framework, each developer/tester has a consolidated profile to which all the associated game mechanics data are recorded and are made available to be viewed by the project stakeholders. It is worth mention that the human feedback can affect 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 an innovative initiation, in this research, we integrate feedback provided by gamification data to be useful to enrich the accuracy of results.

In the maintenance phase, whenever necessary, the traceability information is extracted from the project assemblies (which consist of tests and product code assemblies) to specify the test-to-code traceability links via an automatic Link Retriever component. Then, to interpret and verify the retrieved traceability links, a Link Inspector component is used. This module investigates the status of the traceability links to determine whether they are valid. To validate the status of the traceability links that are embedded as traceability attributes, the respective test case of each traceability link is analyzed. The result of the verification step specifies the links' status. The recovered traceability links are not subjected to verification at the end because they had been incorporated by the test developers in advance. The corresponding information of the links' status is logged to be utilized with a Notification component, which informs users about the links' status. All the retrieved traceability information from Link Inspector component is presented as visualized reports. Web Reporting application and Traceability Console application are provided to present the traceability information in various formats to be helpful for test developers and/or project managers.

IV. IMPLEMENTATION

As a proof-of-concept development, the proposed framework was prototypically implemented (named *GamiTracify*) using Microsoft Visual studio 2013. It is comprised of three main layers, namely Business layer, Service layer and Presentation layer that have been coded using C# programming language. In the Service layer, WCF (Windows Communication Foundation) technology was used. The Presentation layer was implemented using ASP.NET MVC 5 (Model-View Controller) framework that is the Microsoft implementation of MVC design pattern.The prototype contains a dashboard that displays the tracing progress for a project for tracking and managing the project's tracing goals and individuals for motivating team members to create appropriate trace links. The dashboard 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. Personalized views can be created for individual project members.

For **link retriever**, Reflection technique was used. This technique allows known data types, the enumeration of data types in a given assembly, and the members of a given class or value type to be inspected at run-time. Link retriever looks for certain attributes of classes and methods to retrieve test and business classes from the project assemblies. Test methods are recognized with certain attributes like [Test] for NUnit and MbUnit, [TestMethod] for MSTest, [Fact] for xUnit.NET and etc. After detecting test methods, Link Retriever module finds all test cases which have been marked with the proposed traceability attribute, TestToCodeLink. In addition, the names of test methods that have not been marked with the proposed traceability attribute will be recovered as well to be listed and reported to the software development team members. Extracted information will be passed to Link Inspector module. Since this step is executed after each compilation, the required traceability information will be extracted from the latest code to ensure the links always remain up-to-date.

In **link inspector**, to specify the links' status, the stored information in each embedded traceability attribute is used. The embedded information in the traceability attributes are inspected in the body of the test case using static call graph analysis and in the extracted business classes. For each test case, Link Inspector sends the name of related production class embedded in the traceability attribute to Link Retriever in order to get the type information of that production class. The type information contains class members such as properties and methods information. Then, Link Inspector investigates to find an overload method in production code, which agrees with the method called in the test case in relation to the number of parameters and types of parameters. The output of this inspection determines the "valid" or "invalid" status of each link to be utilized by Link Notification.

Link notification reports the retrieved traceability information from Link Inspector in three different presentation formats, which are as following:

- Web Reporting application, which demonstrates links' status as a pie chart and represents the traceability information in grid format with search capability.
- A console application, which helps unit test developers to identify invalid links. Traceability Console application is available for unit test developers to check the traceability information logs in the development environment.
- Integration with visual studio output window to show the traceability link information after each code compilation.

V. Empirical assessment

The purpose of this section is to assess the proposed framework in terms of accuracy to get a full picture of how the performance of this gamified model is correlated with a non-gamified traceability approach, named SCOTCH [26] (which was presented in Section II). We therefore designed our study around the following main research question:

RQ How does the use of the gamified framework (i.e. GamiTracify) impact the ability to have accurate traceability information, compared to the peer approach (i.e. SCOTCH)?

A. Subjects

The subjects participating in the empirical experiment were 18 software development team members with different roles and job experiences in software development as well as testing. The roles of the chosen participants are Software Developer, Software Tester, Team Leader, Scrum Master and Product Owner with up to 20 years of job experience. This group of participants was chosen to investigate the usefulness of the proposed framework from the viewpoint of different roles whom have different job experience levels. This variety of the participants with different roles helps to get better insight of the framework.

B. Case Studies (Objects)

The subjects were asked to perform traceability tasks over three selected software projects: `Dapper`[2], `Log4net`[3] and `Nhibernate`[4]. We chose these projects because the projects exhibit different characteristics. The scales of the selected projects are small, medium and large, respectively, in terms of Cyclomatic Complexity metric and lines of code. The Dapper is an open-source object mapper, lightweight ORM, for .NET and compatible with any database that implements a provider for it. Log4net is an open-source Logging framework for Microsoft .NET Framework. The Logging Services project is intended to provide cross-language logging services for the purpose of application debugging and auditing. Nhibernate is an open-source object/relational mapping tool for .NET environments. The term object/relational mapping (ORM) refers to the technique of mapping a data representation from an object model to a relational data model with a SQL-based schema.

Table II shows the sizes of the selected projects in terms of Cyclomatic Complexity, lines of code, classes, test classes and test cases. Cyclomatic Complexity[5] measures the structural complexity of the code. It is created by calculating the number of different code paths in the flow of the program. This metric for the selected projected was measured by Visual Studio to represent the project size (and not the maintainability factor).

[2] https://github.com/StackExchange/dapper-dot-net
[3] http://logging.apache.org/log4net/
[4] http://nhforge.org/
[5] http://msdn.microsoft.com/en-us/library/bb385914.aspx

TABLE II. CHARACTERISTICS OF THE EXPERIMENTED PROJECTS

Projects	**Characteristics**					
					Unit Tests	
	Size	*Cyclomatic Complexity*	*Lines of Code*	*Number of Classes*	*Number of Test Classes*	*Number of Test Cases*
`Dapper`	Small	867	1087	21	21	86
`Log4net`	Medium	4,023	7,484	235	29	156
`Nhibernate`	Large	39,463	78,645	2576	925	3917

C. *Experimental Design and Process*

The experiment was organized through different sessions. The subjects applied the two different approaches to perform the assigned traceability tasks. The first method, the subjects performed the tasks using *SCOTCH*, while in the second one, they performed the tasks using *GamiTracify*. In order to avoid the "familiarity" with the tasks by participants using two approaches, we switched the order of the tasks to control the effect of this variable on the resulting precision and recall during the sessions. Thus, the experiment was designed in way that of the 18 participants, 9 were randomly assigned to use SCOTCH and the other 9 were chosen to use GamiTracify. This tweak allowed us to minimize the bias in data collection regarding the accuracy by the participants applied in the experiment to investigate the said research question.

Three traceability tasks that participants had been asked to perform, included:

TS1 seeking the test cases associated with two selected methods in Dapper project;

TS2 seeking the test cases associated with two selected methods in Log4net project;

TS3 seeking the test cases associated with two selected methods in Nhibernate project.

To quantify the accuracy, *recall* and *precision* metrics [49] were used. It should be noted that, the better precision implies the lesser percentage of false positives whereas, the better recall implies the higher number of correct links.

To perform the experiment, the three selected projects should have the proposed traceability attributes. Therefore, test-to-code traceability links could be established. As three of the three chosen projects are open-source, the time was limited, and also the scale of the projects was relatively large, a subset of test classes and test cases were randomly selected and few domain experts incorporated the traceability attributes into the selected unit tests. The chosen projects, namely Dapper, Log4net and Nhibernate, consist of 75, 156, 3917 test cases, respectively and the traceability attributes were incorporated into only 75, 147 and 205 test cases of these projects, correspondingly.

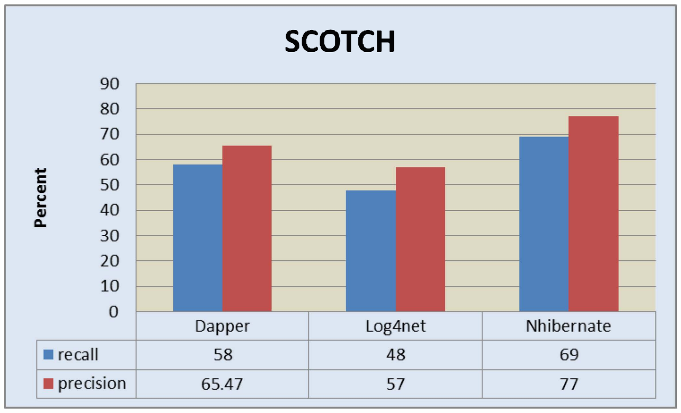
D. *Results*

After the experiment execution, we analyzed the results achieved in this section. The summary of results is shown in Table III. The table shows the *mean* recall and precision of each approach on three object projects.

TABLE III. EXPERIMENTAL DATA

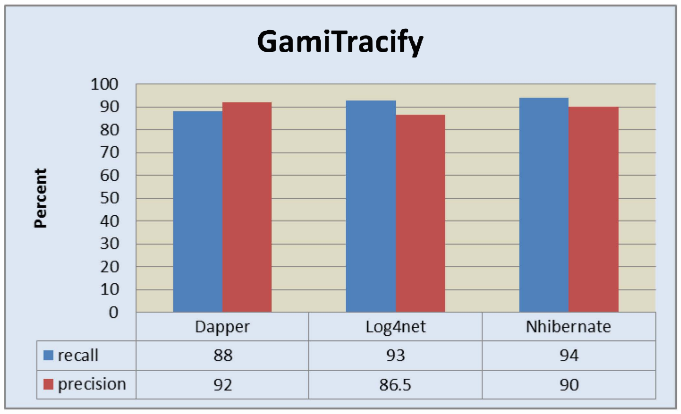
	Recall (%)	**Precision (%)**
SCOTCH	58.33	65.49
GamiTracify	91.66	89.50

The analysis of the achieved results shows that, the GamiTracify gives a considerable better tracing accuracy with less false positives in compared with its peer approach. This matter is evident by the details results related to approaches presented in Fig. 2 and Fig. 3 respectively.



	Dapper	Log4net	Nhibernate
🟦 recall	58	48	69
🟥 precision	65.47	57	77

Fig. 2. The details traceaiblity results realted to SCOTCH



	Dapper	Log4net	Nhibernate
🟦 recall	88	93	94
🟥 precision	92	86.5	90

Fig. 3. The details traceaiblity results realted to GamiTracify

The results overall suggest that gamification can be used in a manner that could have a positive impact on the motivation of the team who used *GamiTracify* to be more effective in contributing to the end-result which is accuracy.

It is expected that results from this evaluation will provide a motivational basis for software enterprises and/or researchers to leverage the proposed framework in their projects.

VI. Conclusion and Future work

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 mainly originate in a lack of motivation and improper engagement of human developers/testers in traceability tasks. This paper proposed an instance solution to alleviate this issue by providing a gamified framework applicable to test-to-code traceability from a human-based perspective. The framework is mainly based on gamification concept to engage developers, which also aims to build a foundation that combines automatic features to retrieve links in later stage of software life cycle, i.e. maintenance.

There will be avenues that can extend the current work. The first one is to extend the custom attribute introduced by the tool to establish traceability links between more software artefacts such as requirements, design, diagrams, and so forth. The second one is to incorporate other relevant software-specific game mechanics, which will be able to provoke the engagement feelings of software engineers involved in their tasks.

Acknowledgment

The author would like to extend his appreciation to New York Institute of Technology for providing support to this work. He also wishes to express his special thanks to all the participants.

References

[1] P. Lago, H. Muccini, and H. v. Vliet, "A scoped approach to traceability management," *Journal of Systems and Software,* vol. 82, pp. 168–182, 2009.

[2] X. Chen and J. Grundy, "Improving automated documentation to code traceability by combining retrieval techniques," in *Proceedings of the 26th IEEE/ACM International Conference on Automated Software Engineering*, 2011, pp. 223-232.

[3] R. Dömges and K. Pohl, "Adapting traceability environments to project-specific needs," *Communications of the ACM,* vol. 41, pp. 54-62, 1998.

[4] S. Winkler and J. V. Pilgrim, "A survey of traceability in requirements engineering and model-driven development," *Software and Systems Modeling,* vol. 9, pp. 529-565, 2010.

[5] R. Watkins and M. Neal, "Why and how of requirements tracing," *IEEE Software,* vol. 11, pp. 104-106, 1994.

[6] A. Ghazarian, "A Research Agenda for Software Reliability," *IEEE Transactions on Reliability, IEEE Reliability Society Technical Operations Annual Technical Report for 2010,* vol. 59, pp. 449-482, 2010.

[7] C.-Y. Huang and S.-Y. Kuo, "Analysis of Incorporating Logistic Testing-Effort Function Into Software Reliability Modeling," *IEEE Transactions on Reliability,* vol. 51, pp. 261-270, 2002.

[8] B. V. Rompaey and S. Demeyer, "Establishing Traceability Links between Unit Test Cases and Units under Test," in *Proceedings of the 2009 European Conference on Software Maintenance and Reengineering*, 2009, pp. 209-218.

[9] H. Ohtera and S. Yamada, "Optimal Allocation & Control Problems for Software-Testing Resources," *IEEE Transactions on Reliability,* vol. 39, pp. 171-176, 1990.

[10] C.-Y. Huang and M. R. Lyu, "Optimal Testing Resource Allocation, and Sensitivity Analysis in Software Development," *IEEE Transactions on Reliability,* vol. 54, pp. 592-603, 2005.

[11] Z. Wang, K. Tang, and X. Yao, "Multi-Objective Approaches to Optimal Testing Resource Allocation in Modular Software Systems," *IEEE Transactions on Reliability,* vol. 59, pp. 563-575, 2010.

[12] P. Kubat and H. S. Koch, "Managing Test-Procedures to Achieve Reliable Software," *IEEE Transactions on Reliability,* vol. R-32, pp. 299-303, 1983.

[13] S.-Y. Kuo, C.-Y. Huang, and M. R. Lyu, "Framework for Modeling Software Reliability, Using Various Testing-Efforts and Fault-Detection Rates," *IEEE Transactions on Reliability,* vol. 50, pp. 310-320, 2001.

[14] A. Espinoza and J. Garbajosa, "A study to support agile methods more effectively through traceability," *Innovations in Systems and Software Engineering,* vol. 7, pp. 53-69, 2011.

[15] R. M. Parizi, S. P. Lee, and M. Dabbagh, "Achievements and Challenges in State-of-the-Art Software Traceability Between Test and Code Artifacts," *IEEE Transactions on Reliability,* vol. 63, pp. 913-926, 2014.

[16] S. K. Sundaram, J. H. Hayes, A. Dekhtyar, and E. A. Holbrook, "Assessing traceability of software engineering artifacts," *Requirements Engineering,* vol. 15, pp. 313-335, 2010.

[17] P. Mader, P. L. Jones, Y. Zhang, and J. Cleland-Huang, "Strategic Traceability for Safety-Critical Projects," *IEEE Software,* vol. 30, pp. 58-66, 2013.

[18] G. Antoniol, G. Canfora, G. Casazza, A. D. Lucia, and E. Merlo, "Recovering traceability links between code and documentations," *IEEE Transactions on Software Engineering,* vol. 28, pp. 970-983, 2002.

[19] A. Marcus and J. I. Maletic, "Recovering documentation-to-source-code traceability links using latent semantic indexing," in *Proceedings of the 25th International Conference on Software Engineering*, Portland, Oregon, 2003, pp. 125-135.

[20] A. Egyed, "A scenario-driven approach to trace dependency analysis," *IEEE Transactions on Software Engineering,* vol. 29, pp. 116-132, 2003.

[21] X. Wang, G. Lai, and C. Liu, "Recovering Relationships between Documentation and Source Code based on the Characteristics of Software Engineering," *Electronic Notes in Theoretical Computer Science,* vol. 243, pp. 121-137, 2009.

[22] A. D. Lucia, F. Fasano, R. Oliveto, and G. Tortora, "Recovering traceability links in software artifact management systems using information retrieval methods," *ACM Transactions on Software Engineering and Methodology,* vol. 16, pp. 13:1-13:50, 2007.

[23] R. Witte, Q. Li, F. F. Informatic, Y. Zhang, and J. Rilling, "Text mining and software engineering: an integrated source code and document analysis approach," *IET Software* vol. 2, pp. 1-19, 2008.

[24] W. Jirapanthong and A. Zisman, "XTraQue: traceability for product line systems," *Software and System Modeling,* vol. 8, pp. 1619-1366, 2009.

[25] A. Bacchelli, M. Lanza, and R. Robbes, "Linking e-mails and source code artifacts," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering*, Cape Town, South Africa, 2010, pp. 375-384.

[26] A. Qusef, G. Bavota, R. Oliveto, A. D. Lucia, and D. Binkley, "Recovering test-to-code traceability using slicing and textual analysis," *Journal of Systems and Software,* vol. 88, pp. 147-168, 2014.

[27] S. Deterding, D. Dixon, R. Khaled, and L. Nacke, "From game design elements to gamefulness: defining "gamification"," in *Proceedings of the 15th International Academic MindTrek Conference: Envisioning Future Media Environments*, Tampere, Finland, 2011, pp. 9-15.

[28] J. Koivisto and J. Hamari, "Demographic differences in perceived benefits from gamification," *Computers in Human Behavior,* vol. 35, pp. 179-188, 2014.

[29] J. a. Fernandes, D. Duarte, C. Ribeiro, C. Farinha, J. a. M. Pereira, and M. M. d. Silva, "iThink : A game-based approach towards improving collaboration and participation in requirement elicitation," *Procedia Computer Science* vol. 15 pp. 66 -77, 2012.

[30] P. Herzig, M. Ameling, and A. Schill, "A Generic Platform for Enterprise Gamification," in *Proceedings of the Joint Working IEEE/IFIP Conference on Software Architecture and 6th European Conference on Software Architecture*, 2012, pp. 219-223.

[31] N. Chen, "GATE: game-based testing environment," in *Proceedings of the 33rd International Conference on Software Engineering*, 2011, pp. 1078-1081.

[32] A. Domínguez, J. Saenz-de-Navarrete, L. de-Marcos, L. Fernández-Sanz, C. Pagés, and J.-J. Martínez-Herráiz, "Gamifying learning experiences: Practical implications and outcomes," *Computers & Education,* vol. 63, pp. 380-392, 2013.

[33] J. Simões, R. D. Redondo, and A. F. Vilas, "A social gamification framework for a K-6 learning platform," *Computers in Human Behavior,* vol. 29, pp. 345-353, 2013.

[34] L. de-Marcos, A. Domínguez, J. Saenz-de-Navarrete, and C. Pagés, "An empirical study comparing gamification and social networking on e-learning," *Computers & Education,* vol. 75, pp. 82-91, 2014.

[35] J. M. Rodríguez Corral, A. Civit Balcells, A. Morgado Estévez, G. Jiménez Moreno, and M. J. Ferreiro Ramos, "A game-based approach to the teaching of object-oriented programming languages," *Computers & Education,* vol. 73, pp. 83-92, 2014.

[36] V. Insley and D. Nunan, "Gamification and the online retail experience," *International Journal of Retail & Distribution Management,* vol. 42, pp. 340-351, 2014.

[37] K. Browne, C. Anand, and E. Gosse, "Gamification and serious game approaches for adult literacy tablet software," *Entertainment Computing,* vol. 5, pp. 135-146, 2014.

[38] S. K. Bista, S. Nepal, C. Paris, and N. Colineau, "Gamification for Online Communities: A Case Study for Delivering Government Services," *International Journal of Cooperative Information Systems* vol. 23, 2014.

[39] D. J. Dubois and G. Tamburrelli, "Understanding gamification mechanisms for software development," in *Proceedings of the 9th Joint Meeting on Foundations of Software Engineering*, Saint Petersburg, Russia, 2013, pp. 659-662.

[40] B. Vasilescu, "Human aspects, gamification, and social media in collaborative software engineering," in *Proceedings of the 36th International Conference on Software Engineering*, Hyderabad, India, 2014, pp. 646-649.

[41] O. Pedreira, F. García, N. Brisaboa, and M. Piattini, "Gamification in software engineering – A systematic mapping," *Information and Software Technology,* vol. 57, pp. 157–168, 2015.

[42] J. Cleland-Huang, O. C. Z. Gotel, J. H. Hayes, P. Mader, and A. Zisman, "Software traceability: trends and future directions," in *Proceedings of the 36th International Conference on Software Engineering*, 2014, pp. 55-69.

[43] A. V. Deursen and L. Moonen, "The video store revisited–thoughts on refactoring and testing," in *Proceedings of 3rd International Conferance, EXtreme Programming and Flexible Processes in Software Engineering*, 2002, pp. 71-76.

[44] H. M. Sneed, "Reverse engineering of test cases for selective regression testing," in *Proceedings of the 8th European Conference on Software Maintenance and Reengineering*, 2004, pp. 69-74.

[45] A. Qusef, R. Oliveto, and A. De Lucia, "Recovering traceability links between unit tests and classes under test: An improved method," in *IEEE International Conference on Software Maintenance (ICSM)*, 2010, pp. 1-10.

[46] A. Rafati, S. P. Lee, R. M. Parizi, and S. Zamani, "A Test-to-Code Traceability Method using .NET Custom Attributes," in *Proceedings of the ACM International Conference on Research in Adaptive and Convergent Systems*, 2015, pp. 489-496.

[47] A. Qusef, G. Bavota, R. Oliveto, A. De Lucia, and D. Binkley, "SCOTCH: Test-to-code traceability using slicing and conceptual coupling," in *Proceedings of the 27th IEEE International Conference on Software Maintenance* 2011, pp. 63-72.

[48] A.Qusef, G. Bavota, R. Oliveto, A. D. Lucia, and D. Binkley, "Evaluating Test-to-Code Traceability Recovery Methods through Controlled Experiments," *Journal of Software: Evolution and Process,* vol. 25, pp. 1167–1191, 2012.

[49] G. Salton and C. Buckley, "Term-weighting approaches in automatic text retrieval," *Information Processing and Management,* vol. 24, pp. 513-523, 1988.