A Metric Framework for the Gamification of Web and Mobile GUI Testing

Filippo Cacciotto, Tommaso Fulcini, Riccardo Coppola, and Luca Ardito
Department of Control and Computer Engineering
Politecnico di Torino
Turin, Italy
first.last@polito.it

***Abstract*—System testing through the Graphical User Interface (GUI) is a valuable form of Verification & Validation for modern applications, especially in graphically-intensive domains like web and mobile applications. However, the practice is often overlooked by developers mostly because of its costly nature and the absence of immediate feedback about the quality of test sequence. This paper describes a proposal for the Gamification of exploratory GUI testing. We define – in a tool and domain-agnostic way – the basic concepts, a set of metrics, a scoring scheme and visual feedbacks to enable a gamified approach to the practice; we finally discuss the potential implications and envision a roadmap for the evaluation of the approach.**

***Index Terms*—Gamification, Software Testing, GUI Testing**

I. Introduction

Evidences from Software Engineering literature suggest that the testing phase is often underestimated by programmers as it is considered a boring, costly and repetitive task, as well as neglected in terms of time and resources [1]. Despite that, testing is a fundamental phase of software development, to avoid error-prone code and application misbehavior, and to reveal defects before release to the final users.

There are many levels and methodologies to conduct software testing, each one with peculiar features, advantages and drawbacks. A crucial testing facet for modern domains is System testing conducted through the Graphical User Interface (i.e., GUI testing), focusing on the visual interaction with the tested apps. In GUI testing, the tester has to interact with the system only via its Graphical User Interface. This allows the tester to perceive the platform in the same way the final user does with the final product. Testing with this approach is important especially when the product to develop is a website or a smartphone app, in which the visual aspect is in continue evolution and most of the interaction with the users is conducted through the GUI itself [2].

Forcing the testers to a visual interaction with the AUT (Application Under Test) allows them to think in a different way compared to what they do when they deal with the code (e.g., in the development of unit tests), and to closely mimic the final user's perception. Despite these benefits, a structured application of this type of testing is often neglected in favor of manual execution of common use cases of the AUT. For these reasons, we envision that Gamification, which consists in the use of elements, philosophies and mechanics that are typical of game design in non-playful contexts [3], can help testers to perceive the testing process in a more entertaining and engaging way, whilst enhancing their performance.

Our idea is to set up a framework useful for building Gamification plugins for GUI testing tools, exploiting gaming concepts that can make the experience of testing more attractive for developers and aid increasing the usage of proper tooling to create, record and execute test sequences and the fun, in order to improve the experience of the tester in place of manual GUI testing.

In this paper we provide: a brief analysis of the Gamification concepts applied to Software Engineering and testing research (section II), the details of the framework we propose (section III), a discussion of the potential impact of our work (section IV) and a roadmap of the prosecutions that we envision for this research (section V).

II. Background and Related Work

In recent years, several attempts have been made to improve the grade of involvement and enjoyment of testers by applying Gamification concepts to the subject.

The main focus up to now has been the application of game elements in the learning process of the subject, applying these new method mainly in academic courses. Evidences of the effectiveness of this method have been reported, especially on the short term, by many researches that applied innovative tools to support the teaching of software testing. [4, 5]

Code Defenders is a tool focused on mutation testing and meant to improve the tester's satisfaction by introducing challenges in the process. Testers take on a challenge by dividing in two teams: the attacker team has to inject bugs and the defenders have to enforce the test suite, trying to predict the newly added bugs. The tool has been applied for an experimentation in an academic course, and the results showed that not only testers were more involved using Code Defenders, but also that the resulting test suite was stronger than both automatically generated one and the one produced without the Gamification tool [5].

Clean Game applies Gamification aspects to the refactoring activity, that is the process of locating and transforming code smells [4]. Code smells are poor design or implementation choices in the source code, that should be rewritten and rethought in favor of more readable and maintainable code. In this game, the tester has to take a multiple choice quiz about each code smell and to recognize them in the code. The experimental usage of this tool using Computer Science students has led them to locate twice as many code smells than without the Gamification concepts.

Other Gamification aspects have been applied to the discipline of Software Engineering in general: FormalZ [6] transforms the formal specification phase into a tower defense game, in which the user has to protect a hypothetical CPU from an hacker attack by submitting the specification given in the correct way. This tool has been subjected to students of a bachelor course of Software Testing that positively perceived the impact of the tool as a help to learn theoretical concepts.

To the best of our knowledge, however, none of the studies about the application of Gamification in the software testing field has had a predominant focus on exploratory GUI testing. The lack of studies about these aspects is the main reason behind our analysis of the context, to highlight the most important aspects to be considered when implementing Gamification in this field.

III. The Proposed Framework

In this section we describe the main concepts on which our metric framework is based, the computed and the adopted Gamification concepts (scoring and visual feedback).

We have developed our first proof-of-concept as an add-on for Scout, a tool for augmented exploratory testing for Web-based applications [7]. We have developed plug-ins to cover all the following aspects also for Android apps, leveraging emulated devices and the Appium tool to obtain details of the shown GUIs.

A. Basic concepts

The main concept on which our set of GUI testing Gamification metrics is based is the *Testing Session*, i.e., the sequence of interactions with the GUI performed by a human tester, emulating a user interaction with the AUT (and thereby, typically starting from the home/main screen).

We represent a testing session with a tree structure, where every node represents a web page and includes some data related to the interactions performed by the tester in that particular page. Each session is uniquely identified by the tester ID and the instant in which it's terminated. This mechanism allows the tool to save each session in a database and to compute both point and aggregate raw data, such as:

- the total number of interactions and widgets encountered by the tester during the session, as well as the amount of the actually interacted ones;
- the total number of pages visited during the session;
- the amount of milliseconds spent on each page and the duration of the entire testing session.

In addition to the quantitative data representing the sessions, we also take into consideration the number of *Issues* and *Easter Eggs* encountered.

Issues represent problems encountered during the exploration of the GUI, either reported manually by the testers with features of the testing tool or errors that are specific of the domain and that can't be recognized automatically (such as broken links, 404 errors, freezes).

Furthermore, we envision the possibility of injecting *Easter Eggs* in the AUT's GUI, i.e. superimposed oval-shaped visual elements, generated after interacting with randomly chose elements. Their purpose is to encourage the tester to explore the domain by opening as many links as possible.

B. Session Metrics

Starting from the raw data described in subsection A, it's possible to compute different metrics that underlie the introduced Gamification elements:

- Page coverage, computed as the percentage of interacted widgets inside a specific page.
- Session coverage, computed as the average page coverage. It represents the total coverage achieved by the tester during the session.
- Total number of interactions per page and average time spent per each interaction.
- Number of widgets and pages encountered for the first time. These numbers can be computed by using a database entry that keeps track of every page (and corresponding interacted widgets) already visited by one or more testers previously.
- Number of reported/encountered bugs during the session.
- Number of easter eggs found during the session.

C. Gamification Elements

1) Final score: The main Gamification element of the proposed framework is a mechanism that assigns a score to each testing session, based on the metrics previously described.

This score allows to evaluate the tester's performance, taking into account different factors, besides introducing a competitive aspect that may encourage testers to put more effort in the testing activity.

The score is composed of two parts: a *base score*, which takes into account the main factors used to evaluate the tester's performance, and a *bonus* one, that allows users to increase their total score and improve their position in the leaderboard.

$$S = S_{base} + S_{bonus}$$

The base score is computed using this formula:

$$S_{base} = a \cdot C + b \cdot EX + c \cdot EF$$

The base score adds up to 100 points and is composed of 3 components weighted by configurable parameters:

- C represents the coverage component and it's computed as the average page coverage reached by the tester during the testing session, according to the formula:

$$C = \frac{\sum\limits_{\forall i \in P} cov_i}{|P|}$$

  where cov_i is the coverage of the i-th page and P is the set of pages visited during the session.

This component is multiplied by default for a coefficient equal to 60%. This is due to the fact that C can be useful to determine how deeply the tester inspected the pages visited during the session.

- EX represents the explorative component of the score and it depends on the percentage of pages visited and widgets interacted for the first time by the current tester, among the other users who have already completed a session before. This component is computed as:

$$EX = \frac{k}{b} \cdot \frac{p_{new}}{p_{tot}} + \frac{h}{b} \cdot \frac{w_{new}}{w_{tot}}, \qquad k + h = b;$$

  where p_{new} and p_{tot} are the newly discovered and the total pages respectively, while w_{new} and w_{tot} are the newly discovered and the total interacted widgets respectively. By default, this component is worth 30% of the total base score; *h* and *k* represent the total base score percentages of each sub-component.
- EF represents the efficiency component and its aim is to determine whether a tester tried to exploit the framework scoring system by clicking on already explored widgets. This component is computed as the ratio between the number of interacted widgets and the total number of interactions registered on the page (including those performed on widgets that had been already clicked by the tester), according to the formula:

$$EF = \frac{w_{hl}}{w_{int}}$$

  By default, this component is worth 10% of the base score.

Testers have the possibility to increase their base score by earning a *bonus score*, that may allow them to reach a higher amount of points. These additional points are computed using the formula:

$$S_{bonus} = d \cdot T + e \cdot P$$

This score is composed of two components and is capped to a maximum of 50% of the base score. The components of the bonus score are kept separated because they depend on aleatory events encountered during the test sequence execution. In particular, the two addends are:

- T, which represents the time component and it's based on the duration of the testing session. Its purpose is to reward longer sessions, in which testers should have explored the GUI more thoroughly (theoretically) than in shorter ones. To avoid any possible exploitation, the metric takes into account the average time spent per each interaction, excluding the sessions where the average time is excessively high (that can be symptom of random clicking) or low (i.e., too much idle time).

  As a consequence, the final formula of the time component is:

$$T = \begin{cases} 0 & s_{int} \leq 2 \ \vee \ s_{int} > 30 \\ 1.5 \cdot t & 2 < s_{int} \leq 5 \\ t & 5 < s_{int} \leq 15 \\ 0.5 \cdot t & 15 < s_{int} \leq 30 \end{cases}$$

TABLE I
GRADES BASED ON FINAL SCORE

Minimum	Maximum	Grade
100	-	S
80	99	A
70	79	B
50	69	C
0	49	D

  where t is the duration of the session (measured in minutes) while s_{int} is the average time spent per interaction (measured in seconds). By default, this component is multiplied by 0.3.
- P is the component related to the problems affecting the AUT that the testers may have encountered during their testing session and it's computed as:

$$P = n_i + n_{ee}$$

  where n_i and n_{ee} are the number of reported issues and the number of the easter eggs found during the session respectively. The default coefficient of this component is equal to 0.2.

Depending on the score obtained, a tester receives an evaluation of its performance, in terms of a grade from D (lowest) to S (highest), according to the ranges in table I.

The usage of these grades is a typical gaming aspect and here we use it as a Gamification element in order to provide the tester with a clear feedback to understand the quality of its testing sequence.

At the end of a session, a tester can visualize a leaderboard where all the testers who performed at least one sequence before are displayed and ordered by the total score they accumulated during the various sessions.

2) Live graphical feedback: To avoid having testers to wait until the end of their session to assess their performance, we have added graphical feedback during the exploration of the GUI (see figure 1). Our proof-of-concept plug-in shows a progress bar on top of each page, representing the percentage of page coverage achieved in real time. Furthermore, if the current page under test has already been visited before by (at least) another tester, the progress bar shows an indicator of the coverage reached before, in order to inform the current tester about the highest score achieved previously on the page. The rationale behind this design choice is to encourage the current tester to achieve higher coverages than those obtained in previous testing sessions.

The tool allows to identify how many widgets have already been interacted, by surrounding them with a proper contour.

Finally, we also envision a live feedback when a page that was never visited previously is discovered, in the form of a star in the corner of the screen. This form of graphical feedback resembles the typical gaming feature of achievements, which are used to keep the player always aware of his progress through the completion of challenges [8].

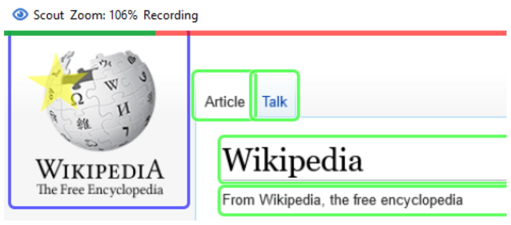
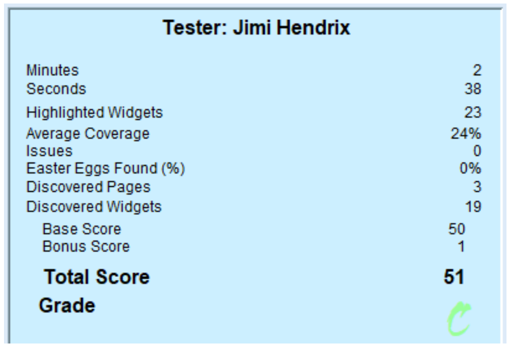

Scout Zoom: 106% Recording
WIKIPEDIA
The Free Encyclopedia
Article
Talk
Wikipedia
From Wikipedia, the free encyclopedia

Fig. 1. Graphical feedback of our proof-of-concept: (a) progress bar showing the coverage on the current page; (b) *star* to indicate a previously undiscovered page; (c) green boxing of interacted widgets.



Tester: Jimi Hendrix

Minutes	2
Seconds	38
Highlighted Widgets	23
Average Coverage	24%
Issues	0
Easter Eggs Found (%)	0%
Discovered Pages	3
Discovered Widgets	19
Base Score	50
Bonus Score	1
Total Score	**51**
Grade	C

Fig. 2. Results screen in our proof-of-concept, showing metrics, score and grade of the session

IV. Potential Impact

Several studies have highlighted the benefits of the application of Gamification to a given technique, especially regarding a better involvement, a higher grade of satisfaction and enjoyment and also better performance when applying the concepts learnt [4, 5, 6].

We expect that the described Gamification can be an useful mean to enhance both tester performance and satisfaction, stimulating a deeper exploration and a more accurate selection of the inputs with the AUT by making the tester challenge his previous efforts and those from other developers. A higher degree of exploration can translate to higher efficiency of test sequences and higher probability of bug and issue discovery, thereby impacting the quality of the AUT.

We envision applications of our proof-of-concept Gamification layer in two different contexts:

- In academic contexts, to enable more engaging teaching of GUI testing in Software Engineering courses, to students that may be more accustomed to gaming aspects;
- In industrial contexts, to take advantage of the higher efficiency and productivity of testers that can be enabled by the challenges introduced by Gamification aspects.

V. Conclusion and Future Work

In this paper we defined a framework to integrate Gamification concepts in GUI testing tools; we implemented a proof-of-concept of the framework as a plugin for an existing exploratory testing tool, named Scout, to apply the concepts to web and mobile applications.

Our immediate future work involves the evaluation of the application of the framework to the creation of test sequences, first in an academic environment (e.g., in a Software Engineering course) and then in an industrial environment. These preliminary evaluations will involve measuring the enjoyment and the engagement of testers (e.g., through surveys) and an inspection of the quality of the proposed framework. This way, it will be possible to gain a preliminary evaluation of human and technical aspects involved in the testing procedures.

We also plan to conduct controlled experiments with a larger and more heterogeneous sample, to evaluate in a quantitative way the efficacy and the quality of the test sequences defined with the gamified approach with respect to test sequences obtained without Gamification.

Finally, other features are also being theorized, such as online competition between testers and an achievement system, the implementation and evaluation of which will follow.

References

[1] M. Beller, G. Gousios, A. Panichella, and A. Zaidman, “When, how, and why developers (do not) test in their ides,” in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, 2015, pp. 179–190.

[2] D. Amalfitano, A. R. Fasolino, P. Tramontana, S. De Carmine, and G. Imparato, “A toolset for gui testing of android applications,” in *2012 28th IEEE International Conference on Software Maintenance (ICSM)*. IEEE, 2012, pp. 650–653.

[3] 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*, 2011, pp. 9–15.

[4] H. M. dos Santos, V. H. Durelli, M. Souza, E. Figueiredo, L. T. da Silva, and R. S. Durelli, “Cleangame: Gamifying the identification of code smells,” in *Proceedings of the XXXIII Brazilian Symposium on Software Engineering*, 2019, pp. 437–446.

[5] G. Fraser, A. Gambi, M. Kreis, and J. M. Rojas, “Gamifying a software testing course with code defenders,” in *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, 2019, pp. 571–577.

[6] W. Prasetya, C. Leek, M. Melkonian, W. Zon *et al.*, “Having fun in learning formal specifications,” in *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*. IEEE, 2019, pp. 192–196.

[7] M. Nass, E. Alégroth, and R. Feldt, “On the industrial applicability of augmented testing: An empirical study,” in *2020 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. IEEE, 2020, pp. 364–371.

[8] E. B. Passos, D. B. Medeiros, P. A. Neto, and E. W. Clua, “Turning real-world software development into a game,” in *2011 Brazilian Symposium on Games and Digital Entertainment*. IEEE, 2011, pp. 260–269.