A Framework for the Gamification of GUI Testing

Chapter 1
A framework for the Gamification of GUI testing

Riccardo Coppola *, Luca Ardito, Tommaso Fulcini
Giacomo Garaccione, Marco Torchiano and Maurizio Morisio

Politecnico di Torino
Department of Control and Computer Engineering
Corso Duca degli Abruzzi, 24
10129, Torino, Italy
name.surname@polito.it
*: corresponding author

Abstract Software testing is a critical activity in the software development process. Several techniques have been proposed, addressing different levels of granularity from low-level unit testing to higher-level exploratory testing through the software's Graphical User Interface (GUI). In modern software development, most test cases are obtained by automated test generation. However, while automation generally achieves high coverage in code-level white-box testing, it does not always generate realistic sequences of interactions with the GUI. By contrast, manual exploratory testing has survived as a costly, error-prone and tedious yet crucial activity. Gamification is seen as an opportunity to increase user satisfaction and engagement while performing testing activities. It could also enable and encourage crowdsourced testing tasks. The purpose of the study described in this chapter is to provide a framework of gamification mechanics and dynamics that can be applied to the practice of manual exploratory GUI testing. We provide an implementation of the framework as an extension of an existing manual exploratory GUI testing for web applications, and we provide a preliminary evaluation of the gamified tool in terms of provided efficiency, effectiveness and User Experience. Our results show that the gamified solution makes the testers obtain test suites with higher coverage, whilst reducing slightly the number of bugs signalled while traversing the applications under test. The gamified tool also was considered to provide a positive user experience and the majority of participants expressed their willingness to use such instruments again in the future. As future work, we foresee the implementation of the framework in a standalone tool and in-depth empirical experiment to evaluate quantitatively the benefits and drawbacks provided by such mechanics in real testing scenarios.

Keywords: Gamification, Software Engineering, Software Testing, Web Application Testing, GUI Testing

1.1 Introduction

Software testing is a critical activity in the software development process. Its main purpose is to detect defects and faults in advance in the code produced, avoiding to release code affected by bugs and security issues, whose repair cost increases once the software is released to the final users. Testing is also utilized to prove conformance to functional requirements and the reliability of software artefacts. Several software testing techniques exist in the literature, ranging from low-level unit testing of atomic components of the software (unit tests) to higher-level exploratory testing through the finalized software (end-to-end or E2E testing). When conducted through the Graphical User Interface (GUI) of the software under test (SUT), E2E testing is typically called GUI testing.

In modern software development test cases are often obtained through automated test generation, a practice that ensures significant time savings and repeatability of test practices. However, while automation generally achieves high coverage in code-level white-box testing activities, automated testing is not always the optimal choice to generate E2E test suites, with the generation of interaction sequences that have a low level of realism in mimicking the final user's interaction with the system. Therefore, a relevant portion of E2E testing activities is still conducted manually by the QA team. Manual Testing is however renowned as a costly, error-prone and tedious yet crucial activity [1]. Throughout this manuscript, we will refer to GUI testing as the tool-aided activity conducted by development companies or external test factories, and not to activities performed by the final users of the applications (e.g., beta testing or crowdsourced verification of web applications). Our focus is also on GUI-based *functional* testing, i.e., to all activities related to the verification of the main features of the SUT.

Gamification, defined as *"the use of game design elements in non-game contexts"* [2], is gaining traction in the latest years in disciplines related to Computer Science, because of its proven capability in motivating, engaging and improving the performance of the participants of tasks to which game mechanics are applied [3] [4]. Several frameworks have been proposed to govern and aid the design of Gamified activities and tools. In this chapter, we adopt as a reference the Octalysis framework proposed by Yu-Kai Chou [5]. The framework identifies eight core drives that represent aspects of human behaviour that can be stimulated through gamification.

Several works in Software Engineering literature have motivated, conceptualized, and evaluated gamification mechanics to improve the results of many activities of the software process [6]. Albeit the primary application of Gamification mechanics is still used primarily in the educational field [7], there is a growing interest by practitioners in implementing them in industrial contexts and tooling [8]. Case studies in the literature have documented encouraging outcomes by such adoption [9].

Of all Software Engineering activities, Software Testing is particularly suitable for the application of the most common Gamification elements. Testing activities, in fact, typically produce quantitative, measurable, and comparable results (e.g., coverage thresholds reached, number of defects found, number of crashes triggered) that can be naturally translated to game-like aspects (e.g., points and leaderboards).

The present chapter has the goal of proposing a structured methodology to apply gamification in the domain of exploratory GUI testing, a facet of the software testing practice which is still not covered by related literature. We propose a set of game mechanics that can be adopted for testing both web and android applications, with a scoring algorithm that can be used in a general context. We aim to chart a viable path for researchers and practitioners that will approach the topic of gamification in the GUI testing discipline.

We report the process of adapting gamified mechanics to the activity of manual exploratory GUI testing, . To that extent, we perform a preliminary investigation of the current state of the art and practice in the field of Gamified Software Testing, to identify the most mentioned tools and adopted mechanics, and the benefits and drawbacks provided by the techniques. We then detail a framework of mechanics for the Gamification of GUI testing, incorporating game elements like session scores, leaderboards, and live graphical feedback. The framework has been developed as a prototype for GUI testing of Web Applications, but it is by design adaptable to different domains. We finally report the findings of an experiment conducted with graduate students, to evaluate the improvements in effectiveness and user experience of the Gamified tool when compared to the non-Gamified equivalent.

The framework we present brings a novel contribution to the current state of the art related to the gamification of software testing, being an example of a gamified tool for manual exploratory GUI testing: an analysis of the literature we have performed has revealed that this is still a relatively unexplored field. More specifically, no solutions have been proposed previously specifically for exploratory manual GUI testing of web applications. The framework extends two previous works: the prototype framework proposed by Cacciotto et al. [10] and an extension of it by Fulcini and Ardito [11], which also saw a first preliminary evaluation of the framework.

The chapter is organized as follows: section 1.2 presents background about Software Testing techniques and Gamification in the Software Engineering discipline. Section 1.3 provides a survey of the existing scientific literature regarding gamified Software Testing. Section 1.4, based on the findings of the literature review, presents a conceptual framework for gamification of manual exploratory GUI testing. Section 1.5 describes the methodology, setting and results of an empirical evaluation of the framework. Section 1.6 discusses the threats to validity of the framework. Finally, section 1.7 concludes the chapter with an overview of the findings and future research directions.

1.2 Background and Related Work

This section illustrates the main concepts of GUI testing: the major available technologies, the main limitations and open challenges. We also introduce background concepts about the utilization of Gamification in Software Engineering.

1.2.1 GUI Testing

GUI Testing is a form of Functional Testing, that exercises a Software Under Test (SUT) of any given domain through its Graphical User Interface. GUI testing exercises the SUT by mimicking the operations that would be performed by its typical end-that In that sense, GUI Testing is a form of End-to-End (E2E) or System testing since it aims at defining scenario-based test cases to cover the functional requirements of the SUT.

Many GUI Testing tools and techniques have emerged in the last three decades and now they are available for different platforms and domains. All methodologies and technologies used for GUI testing share several commonalities. GUI test scenarios are typically defined as a sequence of different *locators*, i.e., graphical elements that have to be recognized and interacted with inside the GUI; each locator in a GUI test sequence is typically associated with a specific operation, e.g., mouse clicks and movements, key presses, or text insertions. GUI test cases also make use of different forms of *oracles*, i.e., visual cues or properties of the GUIs that are checked to verify the conformance of the SUT's behaviour with the functional requirements.

GUI testing activities are not inherently automated; in fact, related literature highlights that a significant portion of GUI testing is still performed manually by practitioners.

Automated support for GUI testing is made available by a large number of commercial and academic tools, that can be classified under two different categorizations. Alégroth et al. define three different *generations* of GUI testing tools, based on the type of *locators* and *oracles used* [12]:

- *1st Generation*, or coordinate-based testing tools, use exact coordinates to locate and verify the presence of elements on the screen. Coordinate-based locators were used in the very first tools in the field, and have been largely abandoned because of their flakiness and unreliability.
- *2nd Generation*, or property-based testing tools, use textual properties of the graphical elements of the GUI to identify and verify them (e.g., XML attributes in layout files of mobile applications, or HTML attributes and properties in DOM files describing the appearance of web apps). Property-based testing tools are currently the most widespread methodology of GUI testing: *Selenium* and *Espresso* are prominent examples of such category of GUI testing tools for the mobile and web domains respectively.

- *3rd Generation*, or visual testing tools, use image recognition to locate and verify images in the pictorial GUI of the application. Visual GUI testing tools are still an emerging research direction in the literature [13].

Another possible categorization of GUI testing techniques can be performed based on the way the sequences of interactions against the GUI are selected and recorded into test scripts [14]:

- *Automation Frameworks and APIs* are tools that allow the creation of scripts involving methods that access the hierarchy of components of the GUI. The frameworks offer methods that allow executing specific operations on the located widgets.
- *Record and Replay* tools allow the tester to manually execute operations against the SUT's GUI to have them recorded into re-executable test scripts.
- *Automated Input Generation (AIG)* tools allow generating the sequences of interactions against the GUI without human scripting. These tools can be based on random input generation or leverage models of the GUI to test.

Although many tools and techniques are available, there is evidence that GUI testing tools are not widely adopted by commercial and open-source projects, and GUI testing is conducted mostly manually, using random tools, or completely neglected.

The scarce diffusion of GUI testing is partly related to inherent technical issues: GUI test cases exhibit very high *fragility* (i.e., the necessity of performing important maintenance operations of GUI test cases when the SUT's GUI evolves [15]), *flakiness* (i.e., the possibility that the same test case produces unpredictable results when applied to the same SUT [16]) and *fragmentation* (i.e., the necessity to execute the test cases against different rendering of the GUI on varying configurations, browsers or devices [17]). Recent surveys in the literature have identified the main limitations of GUI testing in industrial practice. Even if most of the identified challenges are technological and tool-related (e.g., the difficulty in coping with application changes that may break test execution; timing and synchronization issues between the test cases and the SUT; missing means to provide robust identification of GUI widgets), some of the identified challenges are related to the requirement of specific skills and trained professionals for the execution of even trivial GUI testing activities [18].

However, it is widely accepted in related literature that defining test scripts is typically considered by developers as a time-consuming, error-prone and boring activity [19]. The typically low engagement of testing activities makes them often overlooked in computing curricula, thereby creating a lack of knowledge about testing techniques in junior developers [20].

1.2.2 Gamification in Software Engineering

Many works in related literature have applied gamified concepts to the field of Software Engineering.

In their systematic mapping, Pedreira et al. report that Gamification is being adopted in many different phases of the software process [2]. The principal application areas are Software implementation, Project Monitoring and Control, and Collaboration between team members. Software Testing is the fifth most mentioned area in related work about Gamified Software Engineering practices. Gamification in Software Engineering is also a significantly growing trend in the field, as reported in the literature review performed by Barreto et al. [21].

The mentioned surveys identified badges and leaderboards as the most frequently mentioned Gamification aspects used to enhance engagement in Software Engineering activities. The prevalent benefits provided by the application of Gamification are increased performance in performing the gamified activities, higher product quality, and better learning. A few negative effects are also mentioned, e.g. ineffective and eventually dissatisfying experiences, and cognitive overload on the participants.

Gamification is typically applied to Software Engineering practices that require high commitment and cooperation by their participants. de Melo et al., for instance, present a Gamified platform for Version Control systems, to build a ranking of the most active developers contributing to software projects [22]. Ašeriškis et al. describe game rules for a Project Management System that they evaluated through the application of the standard System Usability Scale (SUS). The authors identified benefits provided by the utilization of the platform, whilst guaranteeing relatively high usability to its users [23].

The literature reviews in the Software Engineering discipline report, however, that a high percentage of works in gamification (more than 70%) fail to report practical results obtained by the application of the technique. This aspect underlines the immaturity of the practice in Software Engineering, whilst not taking into account the fundamental aspect of the User Experience provided by the gamified tools and techniques.

1.3 Gamified Software Testing: a State of The Art

This section discusses and analyzes the current applications of Gamification mechanics to GUI testing. The presented results are the outcome of a semi-systematic literature review that was performed on the Google Scholar dataset, by searching for the keywords *gamification* (or *ludicization*, or *gamified*), *software* and *testing*. We then applied a set of inclusion and exclusion criteria, which are reported in Tables 1.1 and 1.2, to the results of this first search, resulting in a total of 43 sources. The execution of a process of backward snowballing on these sources, where we applied the inclusion and the execution criteria to new research items found, followed by a similar process of forward snowballing, resulted in a total of 50 different literature sources about Gamified Software Testing, which we have listed in the online Appendix A[1]. An analysis of this literature review has shown that there is a significant

[1] https://doi.org/10.6084/m9.figshare.21967361.v1

lack of gamified testing tools that focus on exploratory GUI testing: more precisely, the only works we have found that have GUI testing as a focus are the previous works we have based this framework on. We conclude that this is still an unexplored field of research, and we feel that our work can be considered a novelty; we hope that our framework can inspire new research works in the field of gamified GUI testing.

Table 1.1 Inclusion Criteria for the semi-systematic literature review

Inclusion Criteria	Description
IC1	The source is directly related to the topic of gamification applied to the field of Software Testing, is generally defined for the Software Engineering discipline but with clear applicability to the testing activity or explicitly proposes, discusses, improves or applies an approach, a framework, or a prototype related to the gamification of any facet of software testing, including education of software testing subjects.
IC2	The source addresses the topics covered by the review questions.
IC3	The literature item is written in a language that is directly comprehensible by the authors: English, Italian or Chinese.
IC4	The source is an item of white literature with the full text available for download and is published in a peer-reviewed journal or conference proceedings.

Table 1.2 Exclusion Criteria for the semi-systematic literature review

Exclusion Criteria	Description
EC1	The source is not directly related to the topic of gamification applied to the field of software testing.
EC2	The source is not in a language directly comprehensible by the authors.
EC3	The source is an item of white literature, but the full text is not available for download or online reading.
EC4	The source discusses a serious/applied game that cannot be applied to real use case scenarios of the software testing process or software testing education and training.
EC5	The source is not a primary study but a secondary or tertiary study of the topic.
EC6	The source has not been published between 2010 and 2021.

All the collected sources were analyzed to collect the following information, discussed in the subsections below: (i) the most frequently adopted mechanics and

tools for Gamified software testing, and (ii) the benefits and drawbacks of such techniques as discussed in related literature.

1.3.1 Adopted game mechanics

To assess the adoption and diffusion of game mechanics in Gamified Software Testing literature, all the mechanics mentioned in the selected literature underwent a process of *coding*. The codes used were the names of the mechanics in the Octalysis framework. By this analysis, 24 different mechanics of the Octalysis framework were found in the related literature.

In the following, we report the Gamification mechanics that had a number of mentions above the average:

- *Score*: The mechanic belongs to the *Accomplishment* core drive of the Octalysis framework. It implies that the user can earn virtual points after performing specific actions in the gamified system. The score can be used for other game mechanics (e.g., for buying virtual goods). A scoring system is often considered a fundamental building block for a gamified system, as many other dynamics that directly stimulate emotion rely on that. This mechanic was implemented or discussed in 33 different sources.
- *Leaderboards*: The mechanic belongs to the *Accomplishment* core drive of the Octalysis Framework. It consists of comparisons and rankings between the scores obtained by different users of the gamified system. The correlation between score and leaderboard is clear: the former is a mechanic that does not produce emotional value in the user until a competition dynamic, the leaderboard, is stimulated. The mechanic was implemented or discussed in 27 different sources.
- *Levels*: The mechanic belongs to the *Unpredictability* core drive of the Octalysis framework. They consist of the progression of the player through different stages with different objectives. The mechanic was implemented or discussed in 16 different sources.

The complete set of mechanics proposed, along with their definitions, related Octalysis core drive and mentions in the related literature is reported in the online Appendix B of the manuscript[2].

In Figure 1.1 we report the number of mentions for each Gamification core drive defined in the Octalysis Framework. From the graph, it is evident how the main focus of available Gamification implementations for software testing is to provide *Accomplishment* to the users as a positive means of motivation (43 mentions). Conversely, only two sources implemented mechanics related to the *Avoidance* dimension, which is related to the enforcement of correct patterns by applying punishments and maluses to non-conforming users. Few mentions were also gathered by the Gamification mechanics related to the *Epic Meaning* macro category of the Octalysis framework. We

[2] https://doi.org/10.6084/m9.figshare.20425446

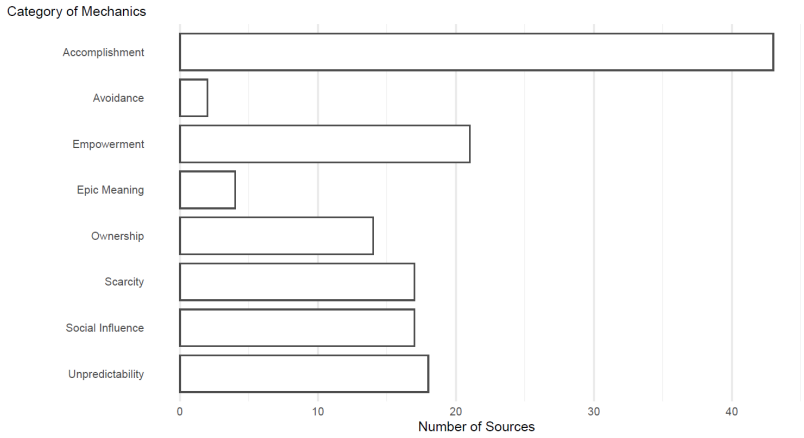

Fig. 1.1 Distribution of mentions in the literature for the core drives of the Octalysis framework

consider such a low number of mentions as an effect of the still prototypical nature of most of the described tools, which did not allow for the implementation of complex narratives.

1.3.2 Gamified Software Testing Tools

In the mined set of literature about Gamified Software Testing, we identify 27 different tools and/or frameworks for Gamified Software Testing. In the online Appendix C[3], we report the full list and tools, providing for each of them a brief description, the adopted mechanics (regarding the Octalysis framework) and the list of literature sources mentioning them.

It is evident from the list of tools that an important focus in Gamified software testing literature is put on software testing education, as gamified mechanics are seen as a primary means of increasing the student's engagement in learning software testing topics. Several Gamified tools aimed at practitioners implement crowd-based mechanisms, to obtain higher coverage and effectiveness (i.e., detected bugs) by generating competition between different testers.

In the following, we report the most mentioned Gamified testing tools in related literature. For each tool, we report the primary source where it has been described and the mechanics that it implements according to the Octalysis framework ; additionally, we present Table 1.3, where we list each tool and the gamified mechanics employed by said tool.

- *CodeDefenders* is a turn-based mutation testing game, in which two players are involved in competitive rounds. One player plays as the attacker, with the

[3] https://doi.org/10.6084/m9.figshare.20425491

objective of injecting faults into the software; the other plays as the defender, with the objective of writing test cases to spot faults. The tool has been originally described by Rojas and Fraser [24] and has originally been used to teach mutation testing in an academic context. In the following years, the tool received further developments introducing more features, along with several related experiments, which enriched the existing literature with experience reports of Code Defenders usage. The tool implements the following Gamification mechanics: duels, scores, leaderboards, puzzles, feedback, and challenges.

- *HALO* is a plugin for Eclipse proposed by Sheh et al. that uses game-like mechanics to make the whole software engineering process more engaging and social [25] [26]. The tool implements an MMORPG-like (Massive Multiplayer Online Role Playing Game) approach to software testing activities. It has been used as the basis for the *Secret Ninja* approach proposed by Kiniry and Zimmerman, in which the Gamification aspects are applied while the users are not aware of their application [27]. The tool implements the following Gamified mechanics: social interaction, quests, storytelling, achievements, levels, and leaderboards.
- *VU-BugZoo*, originally described by Silvis-Cividjian et al. [28], is an educational digital platform to teach software testing, based on a repository of faulty (standalone and embedded) code. The platform engages instructors and learners in a bug-hunting experience which is empowered by the utilization of mechanics typical of game design. The tool implements the trophy and score Gamification mechanics.
- *WReSTT-Cyle* (Web-Based Repository of Software Testing Tools Cyber-Enabled Learning Environments) is a cyber-learning environment that employs several learning and engagement strategies in order to aid the phase of software testing learning. It has been originally described by Clarke et al. [29] as a repository of learning objects to support software testing teaching and has then evolved into different projects named SEP-CyLe (Software Engineering and Programming) and STEM-CyLe (an extension to all STEM disciplines). Originally, the tool covered simple white and black-box unit testing. The cyber-enabled learning environment adopts the following Gamification mechanics: score, leaderboards, badges, timing, levels, rewards, social interaction, and quizzes.
- *Auction-Based Bug Management*: originally described by Usfekes et al. [26], it is a serious game for bug tracking in Application Lifecycle Management Tools. The tool is based on an auction reward mechanism, with the aim of providing an incentive structure for software practitioners to find, resolve and test bugs and malfunctionings of a given SUT. The tool implements the following Gamified mechanics: auctions, virtual goods, timing, badges, and leaderboards.

1.3.3 Advantages and drawbacks of gamification for software testing

On the set of literature items about the Gamification of Software Testing activities, we applied a procedure of *coding* to extract categories of benefits and drawbacks

Table 1.3 Gamified testing tools and their gamified mechanics

Tool	Gamified Elements
CodeDefenders	Duels, Scores, Leaderboards, Puzzles, Feedback, Challenges
HALO	Social Interaction, Quests, Storytelling, Achievements, Levels, Leaderboards
VU-BugZoo	Trophies, Scores
WReSTT-Cyle	Score, Leaderboards, Badges, Timing, Levels, Rewards, Social Interaction, Quizzes
Auction-Based Bug Management	Auctions, Virtual Goods, Timing, Badges, Leaderboards

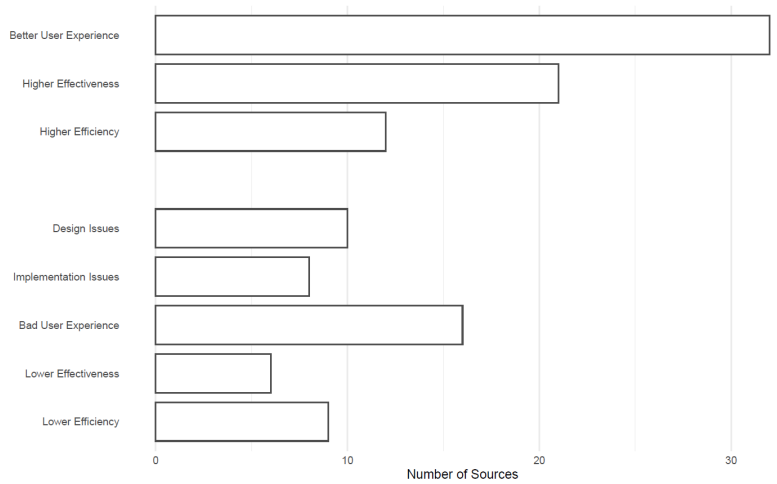

Fig. 1.2 Number of mentions in literature for each category of advantages and limitations

caused by the adoption of gamified mechanics in testing procedures. After the coding was performed, we applied *axial coding* to extract higher-level categories of benefits and drawbacks. We report the full list of advantages and issues extracted from the literature in online Appendix D[4] of this chapter.

The main categories of advantages discussed in related literature are the following:

- **Better User Experience**: Under this category, we include all the benefits related to an increased quality of the user experience provided to the tester when gamified mechanics are applied. 20 sources underline a higher *Engagement* (or *involvement*) guaranteed by game elements in the testing activity, with quantitative empirical results reported by Clegg et al. for unit testing [30] [31].

[4] https://doi.org/10.6084/m9.figshare.20456574

15 different sources highlighted that an important benefit of having game aspects in testing procedures is the guarantee of having more *fun* activities. This aspect was highlighted especially in the educational context [32].
Finally, several Gamification mechanics have been proven to provide additional *Motivation* to the testers involved.

- **Higher Efficiency**. *Efficiency*, as defined by the ISO 9001 standard, is *"the extent to which time, effort or cost is well used for the intended task or purpose"* [33]. Under this category, we include advantages related to reduced efforts and costs in test case definition, generation or execution caused by the application of gamified mechanics. One of the most positively commented aspects of gamified tools is the presence of *Informative Content* in the testing practices, which is able to reduce the effort required by the testers to gather information during the test cases design [34]. Several sources consider Gamification a means to reduce the required effort to perform test-related activities. *Crowdsourced Contributions* is mentioned in several sources and are seen as a primary mean to boost the efficiency of testing procedures [35].
- **Higher Effectiveness**. *Effectiveness*, as defined by the ISO 9001 standard, is *"the extent to which planned activities are realised and planned results are achieved"* [33]. Under this category, we include all the discussed advantages related to an enhancement of the outcomes of gamified testing procedures. Increased effectiveness is measured both in testing education (*Improved Learning*, measured through analysis of grades, [31]) and in testing practice (e.g., increased branch coverage and mutation score in gamified mutation testing, as measured by Fraser et al. [36]). Other specific effectiveness-related aspects are mentioned in other sources, e.g. effectiveness in finding bugs, identifying code smells, finding issues and adding comments to the original code.

The main categories of disadvantages discussed in related literature are the following:

- **Design Issues**. Several studies in the literature report issues related to how the gamified aspects are designed and fit the underlying testing activities. For instance, Garcia et al. and Bryce et al. report cases where gamification interferes with the testing activities, being incompatible with other improvement efforts applied to the testing practice [37] [38]. Five of the collected studies highlight the necessity of proper calibration of gamification mechanics, to disincentivise possible exploits from the users to gain benefits.
- **Implementation Issues**. A limited number of the selected sources mention implementation-related issues for gamified tools. The main concerns in this sense are related to the scalability of gamified approaches, especially the possibility to extend successfully to multiple players the game mechanics that are evaluated on a limited number of subjects [35]).
- **Bad User Experience**. Some reports in the literature report that gamification can make the learning curve for testing procedures steeper [39]. Harranz et al. also report the possibility of *change resistance* for organizations to transition to gamified procedures [40].

- **Lower Effectiveness**. Several sources in the literature report failing attempts at increasing the effectiveness of testing procedures through gamification, both in terms of the bug-finding ability of generated test cases or in learning outcomes [41].
- **Lower Efficiency**. By increasing the concepts that the tester has to learn, Gamification – when not properly designed and applied – can actually add overhead to the testing procedure. Pedreira et al. underline that setting up gamified work environments never has a negligible cost [42]. de Jesus et al. report that gamified environments are inherently complex and require incremental and constant efforts to be built [43].

In Figure 1.2 we report the number of manuscripts in the set of analyzed literature which mention at least one advantage or drawback for each of the defined categories. From the number of mentions, it is evident that the main focus of literature about gamified software testing has a primary focus on the advantages or the drawbacks that are caused to the user experience of the tools and techniques.

1.4 A framework of game mechanics for GUI testing of web apps

In this section, we describe and discuss a framework of gamified mechanics that can be applied to GUI web application testing. To the best of our knowledge, the framework constitutes the first set of gamified elements specifically tailored for the practice of GUI testing. Even if some of the mechanics can be applied to other testing methodologies (e.g., to unit or mutation testing tools), most of the mechanics are specifically intended to aid the definition of second-generation GUI test cases through the Record & Replay methodology.

We have defined two preliminary prototypes of our framework of gamified mechanics: (i) an implementation as a plug-in for the Scout *augmented testing* tool, originally presented by Nass et al. [44]; (ii) a plug-in for the Chrome browser, to enable in-browser generation of test suites for web-applications.

Even though they are specifically implemented for web application testing, the gamified mechanics in the framework are generalizable to GUI testing applied to any software domain (i.e., they can be adapted with little effort to mobile and desktop applications).

Figure 1.3 reports the Octalysis analysis performed for the proposed framework, by assigning one point to each dimension to which the gamified mechanics belong. The following subsections describe the individual gamification mechanics implemented.

1.4.1 Scoring Mechanism and Leaderboard

The principal gamification element of the proposed Gamification framework is a formula to assign a score to each testing session performed by the tester 1.4. The

Fig. 1.3 Octalysis score for the proposed framework

score allows for evaluating the tester's performance, taking into account different factors, whilst introducing a competitive aspect that may encourage testers to put more effort into their activities.

The score is composed of two parts: a *base score*, which takes into account factors that can be used to compare different GUI test sequences on the same working application, and a *bonus score*, which takes into account the bug-finding capability of the developed test suites. As such, the score is calculated by using the following formula:

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

The base score is computed by using the following formula:

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

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

- **Coverage Component** (C): the average page coverage obtained by the tester during the session, according to the formula:

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

  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%.
- **Exploration Component** (EX): a component depending on the percentage of pages visited and widgets interacted for the first time by the current tester. It is therefore computed according to the following formula:

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

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.

- **Efficiency Component** (EF): it is computed as the ratio between the number of interacted widgets and the total number of interactions, thereby according to the formula:

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

By default, this component is worth 10% of the base score. This component aims at measuring the diversity of interactions performed by the tester to avoid exploitations of the scoring mechanism.

The bonus score is computed by using the following formula:

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

The base score adds up to 50 points and is composed of two subcomponents weighted by configurable parameters:

- **Time Component** (T): it is computed on top of the duration of the test session. The rationale for the utilization of a time component is that longer test sequences should allow a more thorough exploration of the GUI. The time component is computed as follows:

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

where t is the duration of the session (in minutes) while s_{int} is the average time spent per interaction (measured in seconds). By default, this component is multiplied by 0.3.

- **Problems Component** (P): it is computed on top of the number of issues reported by the tester during the exploration of the SUT. The default coefficient of this component is equal to 0.2.

1.4.2 Progress Bars

The progress bar is a form of live graphical feedback, that shows the number of widgets that have been interacted with by the tester during the exploration of the SUT.

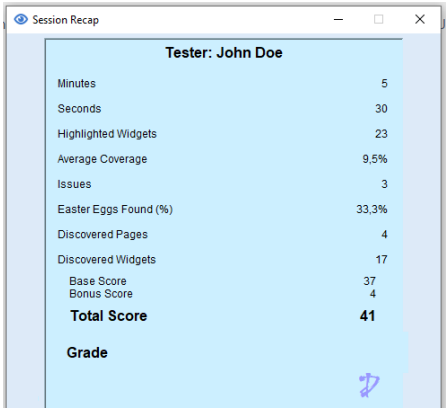

Fig. 1.4 Results screen, showing the metrics measured for the session and the related score

The progress bar is rendered so that a global progress bar shows the percentage of widgets interacted with by the user, in relation to the total amount of widgets present on the page.

For the pages that have already been explored by at least another tester, a blue line is shown on top of the progress bar to indicate the *highest score* in the page, i.e., the maximum coverage reached on such page among all past test sessions.

This element is aimed at providing satisfaction for the progress made by the testers.

1.4.3 Exploration Highlights

Exploration Highlights are a form of live feedback employed to notify that a new page has been discovered by the current tester, i.e. no previous test session has ever visited it.

This element allows informing the tester when their exploration of the SUT is better – in terms of novelty – than previous testers' ones.

1.4.4 Injected Bugs

Injected Bugs are visual modifications that are injected into the AUT's GUI. At the current state of implementation of the prototype, injected bugs are represented as superimposed oval-shaped visual elements, placed over randomly chosen elements in a set of pages of the SUT. The purpose of injected bugs is to encourage the tester to explore the web application by visiting as many pages as possible, whilst providing

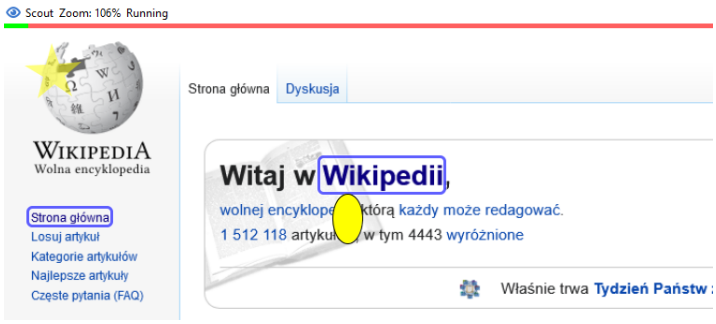

Fig. 1.5 Visual elements shown during the exploration of the SUT: progress bar, exploration highlights, injected bugs

an immediate operation to perform in addition to only recording valid operations over a properly working SUT.

In Figure 1.5 the three visual elements discussed are visible: the progress bar on top of the screen, an exploration highlight in the top-left corner, and an injected bug in the centre of the page.

1.4.5 Achievements

Achievements are used to provide a visual certification that the tester has met specific objectives or requirements.

Achievements are shown through graphical *badges*. The purpose of achievements is to provide gratification to the tester.

Gained achievements are shown as a form of live feedback during the test session in which they are gained. It is possible for a tester to visualize the whole set of achievements gained and those that are still missing.

1.4.6 User profiles and avatars

The mechanic allows defining a profile for each tester registered in the system. The tester can then customize the profile with the preferred avatar, choosing between a set of available ones. Additional items are unlockable by utilizing an in-game virtual currency, which the tester is given every time he unlocks achievements or completes quests or objectives.

Avatars belong – as a mechanic – to the *Ownership* core drive in the Octalysis framework, and are based on the human need to empower their presence and properties.

Fig. 1.6 Profile page of the tester, with the chosen avatar, the achievements' badges and the avatar shop

Each player profile is associated with a certain amount of experience points. As in many Role Playing Game-based Gamification mechanics, experience points allow to increase the player's *level*, and to unlock additional features or customization items when certain levels are reached.

In Figure 1.6 the profile page of the tester is reported. The profile page shows the currently selected avatar, the item shop (where to invest virtual currency to customize the avatar) and the unlocked achievements.

1.4.7 Quests and Challenges

Finally, the gamification framework includes two additional mechanics aimed at encouraging the tester to execute specific actions.

Quests are specific tasks to perform during the execution of testing sequences. They can be tied to specific types of interactions and elements, or to the number of pages visited. The framework considers two different types of quests: daily quests, which are available for a single day; and questlines, proposing a set of pre-defined quests of increasing difficulty. Figure 1.7 shows both types of quests implemented in the framework.

Challenges are specific tasks to be completed that are available only for a set period of time (e.g., for a week). The score of all participating testers is registered for each challenge, and a special leaderboard for the testers competing in a challenge

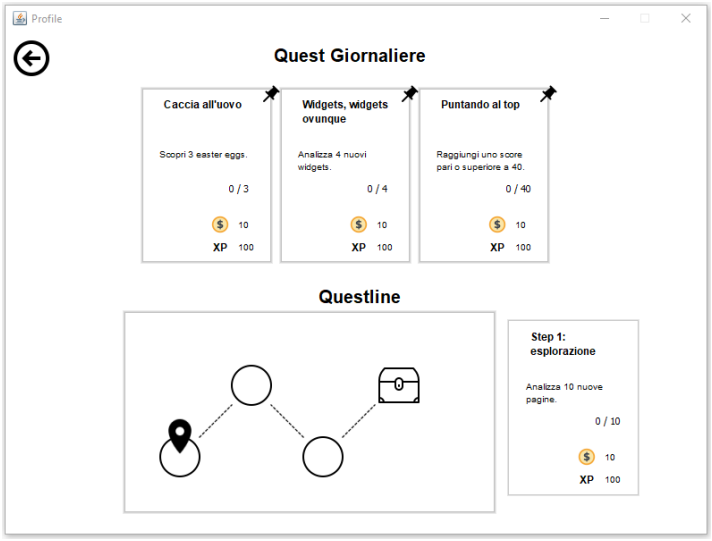

Fig. 1.7 Page displaying quests and challenges, with a section for daily challenges and one for the entire questline, each one having its set of rewards

is created. Prizes can be awarded to the testers that achieve a good placement in the challenge.

Although real-world rewards are indeed a successful extrinsic motivator based on tangible feedback (mostly monetary prizes), a plain usage not supported by a balanced gamified experience would result unappealing and unsustainable in the long term. Providing monetary incentives is an extrinsic motivator, the adoption of which only keeps the user effectively involved in the short term (what is defined as *left-brain* in the Octalysis core drives).

Providing an intrinsic motivating factor such as unpredictability (in the shape of daily quests) is the way we aim at keeping testers engaged with a balanced experience, while attempting to mitigate the over-justification effect, whereby the effect of an extrinsic motivator diminishes as the user becomes accustomed to it.

1.5 Preliminary Evaluation

After the definition of the framework, we performed a preliminary assessment to evaluate the effectiveness and efficiency of the proposed Gamified GUI testing tool.

To that extent, we performed a 2 treatments $\times$ 2 sequences $\times$ 2 objects full factorial (crossover) experiment. The treatment of the experiment was administered as two different versions of the tool: the *Gamified* version (i.e., implementing our framework) and the *Standard* version (i.e., the original version of the Scout tool for exploratory GUI testing of web applications).

Table 1.4 Experiment Design

	Period 1 Object:Treatment	Period 2 Object:Treatment
Group 1	Mezzanine:Gamified	Wagtail:Standard
Group 2	Mezzanine:Standard	Wagtail:Gamified
Group 3	Wagtail:Gamified	Mezzanine:Standard
Group 4	Wagtail:Standard	Mezzanine:Gamified

Table 1.5 Statistics for Coverage and True Positives

		Mezzanine		Wagtail		All	
		S	G	S	G	S	G
Coverage	Mean	8.0%	8.9%	8.6%	10.8%	8.3%	9.9%
	Median	6.2%	7.4%	6.5%	8.8%	6.5%	8.1%
	Std. dev	6.8%	6.0%	5.5%	6.5%	6.2%	6.3%
True positives	Mean	1.87	1.57	3.68	3.58	2.75	2.58
	Median	2.00	1.00	1.00	4.00	2.00	2.00
	Std. dev	1.23	1.06	1.57	1.68	1.67	1.72

The experiment involved 144 participants recruited through convenience sampling among students enrolled in the Software Engineering course held at the Polytechnic University of Turin in the Spring semester of 2021. All participants received both treatments, and received two different tasks to perform, each with a specific subject application. All participants were provided with the task of generating test cases for two different web-based SUTs, manually and utilizing the Scout tool. The applications were selected randomly from a list of open-source applications available in grey-literature [5]. The selected applications were *Mezzanine*[6] and *Wagtail*[7]. The experiment design is reported in Table 1.4.

In our evaluation we focused on three different aspects of the testing practice: effectiveness, efficiency and User Experience. To evaluate *Effectiveness*, we injected artificial bugs in the two experimental objects, and we measured the number of *True Positives*, i.e. bug reports provided by the testers that corresponded to bugs injected in the SUT. To evaluate *Efficiency*, we measured the *coverage* (i.e., the ratio between analyzed widgets and total widgets in the traversed web pages) provided by the generated test cases during the test sequences recorded by the participants. At the end of the experimental sessions, the participants were administered the TAM questionnaire [45], to evaluate the User Experience of the tool. The full TAM questionnaire is reported in an online Appendix E[8].

[5] https://github.com/unicodeveloper/awesome-opensource-apps

[6] https://github.com/stephenmcd/mezzanine

[7] https://github.com/wagtail/wagtail

[8] https://doi.org/10.6084/m9.figshare.20456496

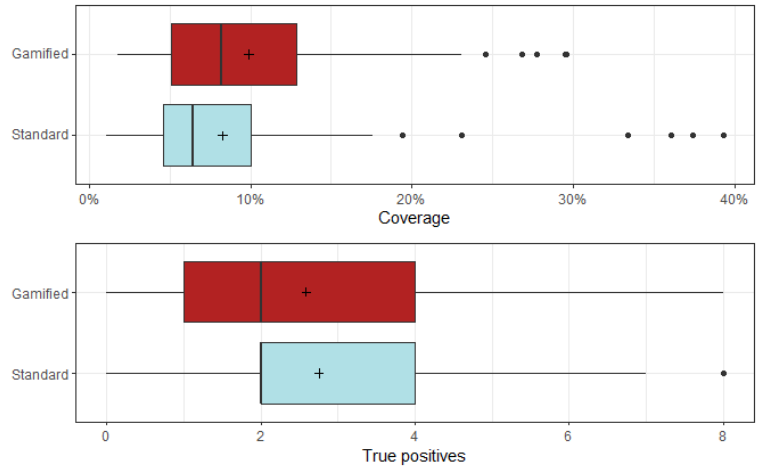

Fig. 1.8 Boxplots for the two metrics observed and measured during the experiment, Average Coverage over pages (Percentage) and True Positives found.

Figure 1.8 shows the boxplots for the distribution of average coverage per page, and the total number of true positives found in each session, aggregated by treatment.

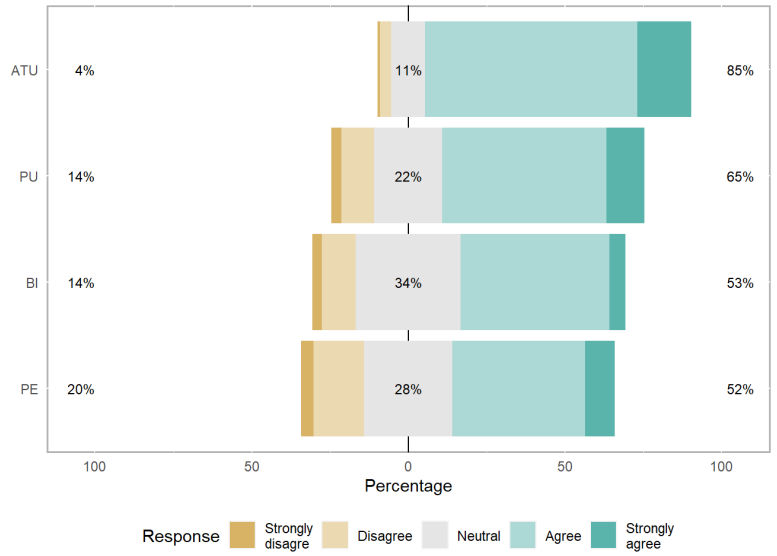

Fig. 1.9 Distribution of the answers (percentage of the responses for each level of the respective Likert scale) provided by the participants to the four categories of the TAM questionnaire: ATU (Attitude Towards Usage), PU (Perceived Usefulness), BI (Behavioural Intention) and PE (Perceived Ease of Use). The results are averaged over all the questions of each category.

The results suggest that the gamified version of the tool achieves higher mean coverage (9.9% against 8.3%), whereas the number of true positives detected was both slightly higher for the standard version of the tool on average (2.75 vs. 2.58).

These results suggest that the inclusion of gamified mechanics primarily caused the participants to focus more on their exploratory testing. Hence, each application page was tested more thoroughly with gamified mechanics in place. We guess that the primary explanation for this behavioural change is connected to the visual highlight features that were added to the tool.

In the experiment, we verified that gamification had a negative (albeit not significant) impact on the defect-finding ability of the testers. We guess that the slightly lower number of defects found on average can be justified by the cognitive overhead introduced by the gamification concepts.

Figure 1.9 reports the distribution of the answers to the TAM Questionnaire. In the graph, we report the mean of the answers for each category of the questionnaire (i.e., Attitude towards Usage or *ATU*, Perceived Usefulness or *PU*, Behavioural Intention or *BI*, Perceived Ease of Use or *PE*).

We observe, on average, that participants had mostly positive perceptions towards all the metrics measured by the TAM model. The metric with the most positive responses was the Attitude towards Usage metric, with 17% of participants strongly agreeing and 66% agreeing that gamification is a desirable addition to the practice of exploratory testing. A high value for the Attitude towards Usage can be considered as a consequence of a positive User Experience of the users in their sessions with the Gamified tool. High positive perceptions were also measured for Perceived Usefulness and Behavioural Intention. These results suggest that most of the participants would use tools with gamification if they had to perform testing activities in the future. Additionally, they indicate that gamification was perceived as valuable and usable. The values for Behavioural Intention and Perceived Usefulness suggest that the Gamified mechanics and aims were easily understandable by the users.

The most negative perceptions were aimed toward Perceived Ease of Use (16% Disagree, 4% Strongly disagree). This result, however, can be justified by the lack of experience of the participants with the practices and the inherently low ease of use of the specific tool that was extended with gamified mechanics.

1.6 Threats to Validity

The potential threats to the framework's validity are discussed according to the categories defined by Wohlin et al. [46].

Threats to Internal Validity concern factors that may affect the results and were not considered in the study. There is no guarantee that the measures selected to evaluate a testing session (coverage, time spent) are the most optimal ones for this purpose: a systematic literature review by Coppola and Alégroth [47] identified 55 different metrics belonging to 4 categories of GUI-based testing: Functional level metrics, GUI level metrics, Model-level metrics, and Code-level metrics. With this

many possible metrics present in the literature for GUI testing, it is not possible to say that the metrics we used for our frameworks are the most beneficial or effective; future experiments and studies are going to be required in order to be able to correctly gauge whether the selected metrics are effective or if they can be replaced with other ones.

Threats to Construct Validity concern the relationship between theory and observation. The framework defines a set of gamified mechanics that have been proven to be effective in multiple studies, but it cannot be assumed that the combination of said mechanics will prove effective for a GUI testing tool, as there is currently no study on gamification of this specific practice. Future experiments are going to be necessary to evaluate whether usage of the tool can lead to improved GUI testing practices, as well as increased interest and motivation for the testers; these experiments will have to evaluate the framework in its entirety as well as the single gamified elements, to identify eventual weak links.

Threats to External Validity concern whether the results can be generalized. We cannot affirm for certain that the tool will prove to be effective for all cases of exploratory GUI testing of web applications, mainly due to two factors: the selection of gamified mechanics, which may appear to be effective on paper but then prove itself not optimal when applied to real-world scenarios, and the absence of other studies and experiments on gamified GUI testing that can be compared to the framework. The fact that this framework consists of, to the extent of our knowledge, a novelty for the current literature, means that we cannot consider our choices and methods to be generalized for all possible facets of exploratory GUI testing. There is also the risk of having selected gamified elements that cannot be applied to every possible kind of web application or to different domains; different web development strategies may not interact correctly with the framework, for example, and this means that the defined strategy cannot be generalized to the entire field of GUI testing for web applications.

1.7 Conclusion and Future Directions

In this chapter, we have described a framework of gamified mechanics to be applied to exploratory GUI testing of web applications. To our best knowledge, the framework constitutes the first effort in adapting gamification mechanics to GUI testing. We complement the definition of the framework with a literature review of Gamification applied to Software Testing, to provide a view of the current state of the art about gamified testing tools, along with the most utilized mechanics, and the mentioned advantages and drawbacks of the technique.

The framework contains seven different mechanics. Three of them represent novel contributions w.r.t. the existing literature: a scoring mechanism specifically defined for GUI Testing of web applications, and graphical feedback (exploration highlights, and progress bars) that are specifically designed to follow the typical procedure of exploratory testing of web-based SUTs.

We have implemented the mechanics in a prototype tool, developed as both a plug-in for an existing Augmented testing tool (Scout) and as a plug-in for the Chrome browser. The implementation of the framework allowed us to perform a preliminary evaluation, in the form of a controlled experiment with 144 participants.

The performed experimentation showed that gamified mechanics provide higher page coverage than non-gamified GUI testing. From a bug-finding standpoint, gamification has shown to have no significant beneficial or detrimental effects. Finally, gamification was considered to provide a good user experience. Our experiment thereby confirms that, with no loss in effectiveness of the generated test sequences, gamification can enhance the User Experience provided to software testers. A better user experience can lead to higher productivity, engagement and quality of test cases produced by testers. The conduction of more well-structured empirical experimentations, including more metrics to verify effectiveness, efficiency and user experience, and the consequent evaluation of the interaction between the different measures, will be crucial to provide a dependable assessment of the benefits provided by Gamification. The current positive effects on coverage can in fact be considered as direct behavioural consequence of some introduced gamification mechanics (e.g., progress bar and visualization highlights) with negligible or even detrimental effects on the real quality of generated test cases with the gamified tool.

At the current state of implementation, the tool is still in a prototypal state and it needs to be deployed on the tester's machine to work; in our future development, we foresee a distributed implementation that will allow the utilization of the tool with crowd-testing purposes, and effective utilization of competitive mechanics (e.g., leaderboards).

Future research directions include the adoption of the tool in an educational context – e.g., a Software Engineering course – in order to conduct a longitudinal study to evaluate the effects of utilizing Gamification when teaching System-level and GUI testing to students. Moreover, we believe it is important to evaluate the impact of the individual proposed mechanics in isolation.

References

1. E. Borjesson, R. Feldt, Automated system testing using visual gui testing tools: A comparative study in industry, in: 2012 IEEE Fifth International Conference on Software Testing, Verification and Validation, IEEE, 2012, pp. 350–359.
2. O. Pedreira, F. García, N. Brisaboa, M. Piattini, Gamification in software engineering–a systematic mapping, Information and software technology 57 (2015) 157–168.
3. M. V. Mäntylä, K. Smolander, Gamification of software testing-an mlr, in: International conference on product-focused software process improvement, Springer, 2016, pp. 611–614.
4. L. Rodrigues, F. D. Pereira, A. M. Toda, P. T. Palomino, M. Pessoa, L. S. G. Carvalho, D. Fernandes, E. H. Oliveira, A. I. Cristea, S. Isotani, Gamification suffers from the novelty effect but benefits from the familiarization effect: Findings from a longitudinal study, International Journal of Educational Technology in Higher Education 19 (1) (2022) 1–25.
5. Y.-k. Chou, Actionable gamification: Beyond points, badges, and leaderboards, Packt Publishing Ltd, 2019.

6. C. Hosseini, O. Humlung, A. Fagerstrøm, M. Haddara, An experimental study on the effects of gamification on task performance, Procedia Computer Science 196 (2022) 999–1006, international Conference on ENTERprise Information Systems / ProjMAN - International Conference on Project MANagement / HCist - International Conference on Health and Social Care Information Systems and Technologies 2021.
URL https://www.sciencedirect.com/science/article/pii/S1877050921023255
7. C. Wang, J. He, Z. Jin, S. Pan, M. Lafkihi, X. Kong, The impact of gamification on teaching and learning physical internet: a quasi-experimental study, Industrial Management & Data Systems 122 (2022) 1499–1521.
8. M. L. Jensen, R. T. Wright, A. Durcikova, S. Karumbaiah, Improving phishing reporting using security gamification, Journal of Management Information Systems 39 (3) (2022) 793–823.
9. O. Liechti, J. Pasquier, R. Reis, Supporting agile teams with a test analytics platform: a case study, in: 2017 IEEE/ACM 12th International Workshop on Automation of Software Testing (AST), IEEE, 2017, pp. 9–15.
10. F. Cacciotto, T. Fulcini, R. Coppola, L. Ardito, A metric framework for the gamification of web and mobile gui testing, in: 2021 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW), 2021, pp. 126–129.
11. T. Fulcini, L. Ardito, Gamified exploratory gui testing of web applications: a preliminary evaluation, in: 2022 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW), 2022, pp. 215–222.
12. E. Alégroth, Z. Gao, R. Oliveira, A. Memon, Conceptualization and evaluation of component-based testing unified with visual gui testing: an empirical study, in: 2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST), IEEE, 2015, pp. 1–10.
13. L. Ardito, A. Bottino, R. Coppola, F. Lamberti, F. Manigrasso, L. Morra, M. Torchiano, Feature matching-based approaches to improve the robustness of android visual gui testing, ACM Transactions on Software Engineering and Methodology (TOSEM) 31 (2) (2021) 1–32.
14. M. L. Vasquez, K. Moran, D. Poshyvanyk, Continuous, evolutionary and large-scale: A new perspective for automated mobile app testing, arXiv preprint arXiv:1801.06267 (2018).
15. R. Coppola, M. Morisio, M. Torchiano, Mobile gui testing fragility: a study on open-source android applications, IEEE Transactions on Reliability 68 (1) (2018) 67–90.
16. A. M. Memon, M. B. Cohen, Automated testing of gui applications: models, tools, and controlling flakiness, in: 2013 35th International Conference on Software Engineering (ICSE), IEEE, 2013, pp. 1479–1480.
17. M. Kamran, J. Rashid, M. W. Nisar, Android fragmentation classification, causes, problems and solutions, International Journal of Computer Science and Information Security 14 (9) (2016) 992.
18. M. Nass, E. Alégroth, R. Feldt, Why many challenges with GUI test automation (will) remain, Information and Software Technology 138 (2021) 106625.
19. P. S. Kochhar, F. Thung, N. Nagappan, T. Zimmermann, D. Lo, Understanding the test automation culture of app developers, in: 2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST), IEEE, 2015, pp. 1–10.
20. D. E. Krutz, S. A. Malachowsky, T. Reichlmayr, Using a real world project in a software testing course, in: Proceedings of the 45th ACM technical symposium on Computer science education, 2014, pp. 49–54.
21. C. F. Barreto, C. França, Gamification in software engineering: A literature review, in: 2021 IEEE/ACM 13th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE), IEEE, 2021, pp. 105–108.
22. A. A. de Melo, M. Hinz, G. Scheibel, C. Diacui Medeiros Berkenbrock, I. Gasparini, F. Baldo, Version control system gamification: A proposal to encourage the engagement of developers to collaborate in software projects, in: G. Meiselwitz (Ed.), Social Computing and Social Media, Springer International Publishing, 2014, pp. 550–558.
23. D. Ašeriškis, R. Damaševičius, Gamification of a project management system, in: The Seventh International Conference on Advances in Computer-Human Interactions, 2014, pp. 200–207.

24. R. M. Parizi, On the gamification of human-centric traceability tasks in software testing and coding, in: 2016 IEEE 14th International Conference on Software Engineering Research, Management and Applications (SERA), 2016, pp. 193–200.
25. J. Ruohonen, L. Allodi, A bug bounty perspective on the disclosure of web vulnerabilities, ArXiv abs/1805.09850 (2018).
26. Ç. Üsfekes, E. Tüzün, M. Yilmaz, Y. Macit, P. M. Clarke, Auction-based serious game for bug tracking, IET Softw. 13 (2019) 386–392.
27. J. Bell, S. Sheth, G. Kaiser, Secret ninja testing with halo software engineering, in: Proceedings of the 4th International Workshop on Social Software Engineering, SSE '11, Association for Computing Machinery, New York, NY, USA, 2011, p. 43–47.
28. N. Silvis-Cividjian, R. Limburg, N. Althuisius, E. Apostolov, V. Bonev, R. Jansma, G. Visser, M. Went, Vu-bugzoo: A persuasive platform for teaching software testing, in: Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education, ITiCSE '20, Association for Computing Machinery, New York, NY, USA, 2020, p. 553.
29. Y. F. P.E., P. J. Clarke, Gamification-based cyber-enabled learning environment of software testing, in: 2016 ASEE Annual Conference & Exposition, ASEE Conferences, New Orleans, Louisiana, 2016, pp. 1–16, https://peer.asee.org/27000.
30. B. S. Clegg, J. M. Rojas, G. Fraser, Teaching software testing concepts using a mutation testing game, in: 2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering Education and Training Track (ICSE-SEET), 2017, pp. 33–36.
31. Y. Sun, Design and implementation of a gamified training system for software testing, Advances in Education 10 (2020) 395–400.
32. G. Fraser, A. Gambi, M. Kreis, J. M. Rojas, Gamifying a software testing course with code defenders, in: Proceedings of the 50th ACM Technical Symposium on Computer Science Education, SIGCSE '19, Association for Computing Machinery, New York, NY, USA, 2019, p. 571–577.
33. I. 9001:2005, Quality management systems - requirements, Standard, International Organization for Standardization (2005).
34. B. Lőrincz, B. Iudean, A. Vescan, Experience report on teaching testing through gamification, in: EASEAI 2021, Association for Computing Machinery, New York, NY, USA, 2021, p. 15–22.
35. S. Amiri-Chimeh, H. Haghighi, M. Vahidi-Asl, K. Setayesh-Ghajar, F. Gholami-Ghavamabad, Rings: A Game with a Purpose for Test Data Generation, Interacting with Computers 30 (1) (2017) 1–30.
36. G. Fraser, A. Gambi, J. M. Rojas, Teaching software testing with the code defenders testing game: Experiences and improvements, in: 2020 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW), 2020, pp. 461–464.
37. R. Bryce, Q. Mayo, A. Andrews, D. Bokser, M. Burton, C. Day, J. Gonzolez, T. Noble, Bug catcher: A system for software testing competitions, in: Proceeding of the 44th ACM Technical Symposium on Computer Science Education, SIGCSE '13, Association for Computing Machinery, New York, NY, USA, 2013, p. 513–518.
38. F. García, O. Pedreira, M. Piattini, A. Cerdeira-Pena, M. Penabad, A framework for gamification in software engineering, Journal of Systems and Software 132 (2017) 21–40.
URL https://www.sciencedirect.com/science/article/pii/S0164121217301218
39. K. Berkling, C. Thomas, Gamification of a software engineering course and a detailed analysis of the factors that lead to it's failure, in: 2013 International Conference on Interactive Collaborative Learning (ICL), 2013, pp. 525–530.
40. E. Herranz, R. Colomo-Palacios, A. Al-Barakati, Deploying a gamification framework for software process improvement: Preliminary results, in: J. Stolfa, S. Stolfa, R. V. O'Connor, R. Messnarz (Eds.), Systems, Software and Services Process Improvement, Springer International Publishing, Cham, 2017, pp. 231–240.
41. G. M. de Jesus, L. N. Paschoal, F. C. Ferrari, S. R. S. Souza, Is it worth using gamification on software testing education? an experience report, in: Proceedings of the XVIII Brazilian Symposium on Software Quality, SBQS'19, Association for Computing Machinery, New York, NY, USA, 2019, p. 178–187.

42. O. Pedreira, F. García, M. Piattini, A. Cortiñas, A. Cerdeira-Pena, An architecture for software engineering gamification, Tsinghua Science and Technology 25 (6) (2020) 776–797.
43. G. M. d. Jesus, F. C. Ferrari, L. N. Paschoal, S. d. R. S. d. Souza, D. d. P. Porto, V. H. S. Durelli, Is it worth using gamification on software testing education? an extended experience report in the context of undergraduate students, Journal of Software Engineering Research and Development 8 (2020) 6:1 – 6:19.
URL https://sol.sbc.org.br/journals/index.php/jserd/article/view/738
44. M. Nass, E. Alégroth, R. Feldt, Augmented testing: Industry feedback to shape a new testing technology, in: 2019 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW), IEEE, 2019, pp. 176–183.
45. Y. Lee, K. A. Kozar, K. R. Larsen, The technology acceptance model: Past, present, and future, Communications of the Association for information systems 12 (1) (2003) 50.
46. C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, A. Wesslén, Experimentation in software engineering, Springer Science & Business Media, 2012.
47. R. Coppola, E. Alégroth, A taxonomy of metrics for gui-based testing research: A systematic literature review, Information and Software Technology 152 (2022) 107062.
URL https://www.sciencedirect.com/science/article/pii/S0950584922001719