A Gaming Quest to Improve Web Locators Robustness

Diego Clerissi
University of Genova
Genova, Italy
diego.clerissi@dibris.unige.it

Maurizio Leotta
University of Genova
Genova, Italy
maurizio.leotta@unige.it

Filippo Ricca
University of Genova
Genova, Italy
filippo.ricca@unige.it

Abstract

Web applications have become fundamental to our lives, thus testing methods ensuring their quality are essential. Usually, Web test automation frameworks and tools use locators to interact with the GUI, serving as hooks to the widgets within Web pages. However, locators are known to be one of the most significant points of fragility in Web testing, due to their susceptibility to rapid software evolution impacting the Web pages structure. While gamification strategies have been recently incorporated into the Web testing process to increase testers engagement, by means of tasks and rewards tailored around test activities, they have not yet been applied to locators robustness. In this paper, we introduce TestQuest, a tool under development that adapts gamification to improve locators robustness, by employing evaluation metrics and best practices designed to make testing via locators more engaging for testers, potentially enhancing the overall test quality.

CCS Concepts

• **Software and its engineering** → **Software verification and validation**; **Development frameworks and environments**.

Keywords

Gamification, Web Testing, Locator, Page Object, Test Robustness.

ACM Reference Format:
Diego Clerissi, Maurizio Leotta, and Filippo Ricca. 2024. A Gaming Quest to Improve Web Locators Robustness. In *Proceedings of the 3rd ACM International Workshop on Gamification in Software Development, Verification, and Validation (Gamify '24), September 17, 2024, Vienna, Austria.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3678869.3685684

1 Introduction

The widespread use of Web applications in our lives requires effective testing techniques to ensure the test artifacts remain aligned with the rapid software evolution [25]. One of the most widely used Web testing tools is Selenium [41], which provides APIs to interact with Web elements via locators, i.e., hooks pointing to elements according to several strategies operating over the Document Object Model (DOM) describing the Web pages (e.g., locating an element by name or by XPath descriptor).

Writing test cases manually, even with the aid of test-assistance tools, can be a burdensome task. Automated test generation techniques can be employed, from basic random exploration strategies to more sophisticated ones, involving systematic explorations of Web pages, scriptless approaches driven by GUI changes, and reinforcement learning algorithms [8]. However, these techniques often rely on locators, which are found to be one of the main points of fragility in Web testing [22], as they rely on the structural properties of the DOM, meaning that even small changes on the GUI during software evolution may potentially break the links between locators and Web elements. Ensuring that locators are robust enough to survive software evolution, and repairing them when they fail, requires significant effort in identifying the most robust locator strategies, making it a time-consuming and tedious task. There exist techniques to automatically generate locators that incorporate robustness heuristics, but these may still require iterative manual fixtures to adapt them to the application under test (e.g., to implement parametric test scripts) [28, 30, 33].

In support of Web testing, the Page Object was proposed as a powerful design pattern [15, 24], whose main contribution is placing an indirection layer between test logic and Web pages structure, making test scripts agnostic of any structural details, encapsulating them within page abstractions. This intermediate layer can ensure that changes in the Web pages require minimal updates to the test code, as the locators are centralized within Page Objects, thus favoring more robust and maintainable test artifacts. Still, manual testing activities survive as Page Objects can require ad-hoc repair interventions and design decisions.

The gamification paradigm has thus emerged to adapt game elements even to the testing domain, to make testing activities more appealing [10]. Gamification may help in promoting testing so that it is performed more frequently, tailoring tasks and rewards around patterns and best practices to guide the testers, and challenge their creativity in producing test artifacts of high quality. Additionally, gamification can be used for educational purposes, conveying testing concepts [4]. However, in the literature, unit testing is the main focus of gamification, whereas Web and GUI testing have mostly been neglected [10, 17].

To this end, we propose TestQuest, a plugin for IntelliJ IDEA that supports locators robustness in Web testing, by integrating a gamification infrastructure within the test environment, thus encouraging the testers to implement robust locators in exchange for rewards, achievements, and profile customization. To the best of our knowledge, gamification has not yet been applied in the context of locators robustness. This paper is organized as follows: Section 2 provides some background and literature review about Web testing and gamification domains, then Section 3 presents the TestQuest tool, Section 4 discusses the tool limitations and advancements, and finally Section 5 drafts some conclusions.

Gamify '24, September 17, 2024, Vienna, Austria

ACM ISBN 979-8-4007-1113-8/24/09
https://doi.org/10.1145/3678869.3685684

2 Background

In the following, we introduce some literature background about locators, Page Object pattern, and gamification, taken as inspiration for our approach. As an example to guide the presentation of the key concepts, Figure 1 shows a Selenium test script accompanied by the login Web page of the open-source application PrestaShop[1], subject of the example. The test script uses a DRIVER object provided by Selenium to manage browser instances (e.g., Firefox). The first line of the LOGINTEST() method is a call to the Selenium API that opens the app at the specified URL. The next three instructions retrieve the widgets to interact with (two text fields for email and password, respectively, and submit button), using the FINDELEMENT() method, which supports various locator strategies over the DOM, through the BY class; once located, the SENDKEYS() and CLICK() commands are executed on the widgets to complete the form submission, respectively by providing the input data to fill the text fields and by clicking on the submit button.

2.1 Locator

In Web testing, a locator is a hook pointing to a Web element within a Web page. A locator is characterized by a *type*, referring to the strategy exploiting the Web page DOM property to locate a Web element, and a *value*, determining the value expected for that DOM property. In the context of Selenium, locators are the basic tiles of the BY class[2]. In the example of Figure 1, locator types are *id* and *name*, whereas locator values are "email", "passwd", and "submitLogin". For instance, the email text field is retrieved by searching for a Web element having "email" as *id* in the DOM, see Figure 2.

The existing locator strategies are many. Selenium supports localization by *id*, by *name*, by *className*, by *linkText*, by *CSS* (Cascading Style Sheets), by *tagName*, and by *XPath* [41]. According to the literature, the most robust locator strategy is the one based on *id*s [25, 26], typically unique and rarely subject to changes. However, identifiers can be onerous to set for each Web element within the DOM, thus they are often auto-generated or set for the most relevant Web elements only, making the strategy not applicable to all cases. On the other hand, since XPaths exploit the DOM structure, they are always applicable, with caution.

The XPath strategy allows elements to be located through expressions pointing to them with respect to the page structure: *absolute* XPaths (i.e., expressions considering all tag levels from the */html* root element within the DOM to the target element) should be avoided as they are very fragile, in favor of *relative* XPaths, that rely on fewer attributes. In Figure 3, two XPaths pointing to a text field are shown. The absolute XPath (above) exposes the entire structural path to the widget, making it more fragile in case of GUI evolution (e.g., when a element within the page is removed), whereas the relative XPath (below) is more robust as it refers to fewer structural elements.

As locators are a well-established foundation for automated Web testing, they are also known as the primary cause of test fragility [22], thus requiring the tester to perform a tedious and costly manual repair activity when they break. In literature, there exist techniques to support the tester in such activity, including (1) automatic generation of robust locators inspired by heuristics and qualitative metrics, (2) automatic repair of broken locators, and

[1]https://prestashop.com/
[2]https://www.selenium.dev/selenium/docs/api/java/org/openqa/selenium/By.html

Figure 1: A Selenium login test script (left) and the corresponding login Web page under test (right).

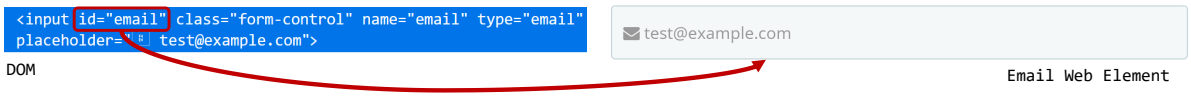

Figure 2: A Locator (left) and the corresponding Web element (right).

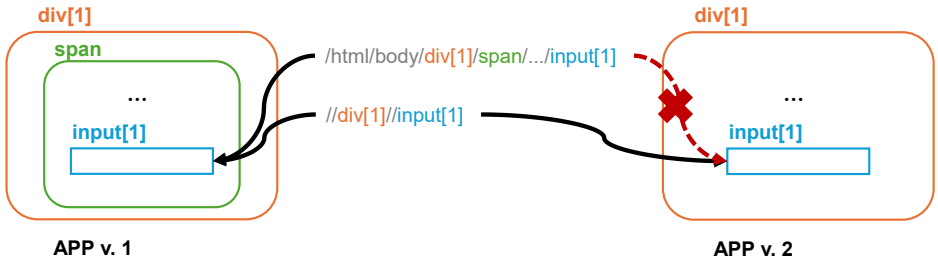

Figure 3: Absolute (above) versus relative (below) XPaths on two app versions.

(3) Web application enrichment with data to improve test robustness [11]. In 2022, Fulcini *et al.* [18] proposed a linter, as a support for GUI testing, designed to detect test smells and antipatterns in test artifacts, partially addressing the locators robustness problem (e.g., avoid using absolute XPaths). ROBULA+ [30] implements refinement transformations to produce robust XPaths according to a set of heuristics (e.g., when building XPaths, the *name* DOM property is prioritized over the *src* property, since *src* might expose the Web application directory structure). Later, Leotta *et al.* extended ROBULA+ XPath generation adopting a more sophisticated statistical adaptive algorithm instead of ad-hoc heuristics [28]. Another interesting solution is Similo [32]. Nguyen *et al.* [33] propose XPaths building heuristics based on DOM elements neighborhood rather than just properties about target elements. Even though these techniques might be used in conjunction with a multi-locator strategy [29] (i.e., each web element is associated with multiple different locators) to lead to robust test suites, the auto-generated locators may still be weak in terms of readability, and might require either high computational cost to be produced or iterative manual adjustments to adapt them to the application under test (e.g., to implement parametric test scripts). VISTA automatic repair tool [38] extracts information from test executions, to identify GUI changes that may cause test breakages between Web app versions. In the seminal approach WATER [7], later on extended by WATERFALL [21], Levenshtein distance is used as a measure to identify XPaths most likely subject to breakages across multiple application versions. COSER [5] approach has been proposed to address larger GUI changes on Mobile apps and support the repair of failing tests, by capturing internal (e.g., code snippet defining functionalities) and external semantics (e.g., graphical appearance properties) about GUI elements that might match between multiple app versions. Di Meglio and Starace [13] elaborate fragility score metrics to predict End-to-End Web test breakages, based on comparisons between source code of Web tests and of Web pages of multiple app releases. Fasolino *et al.* [14] propose a technique to enhance the Web app source code with behavior-agnostic hooks to allow unique Web element localization, enforcing test case robustness without affecting the code semantics. LED [1] tool guides the tester in adding identifiers to Web elements during Web browsing sessions anticipating testing, thus improving locators robustness for Web elements not originally designed with them.

All these techniques can fail if a drastic application evolution occurs, requiring manual effort to define new locators or to repair existing ones, and high computational cost when automation steps are instead employed. Additionally, some of these works need multiple versions of the application under test or the access to the source code to be fully available.

2.2 Page Object Pattern

The Page Object is an established End-to-End design pattern for Web testing, well integrated in test automation with Selenium [15, 24]. Technically speaking, a Page Object is an Object Oriented class structurally representing a Web page, which translates Web elements and functionalities into attributes and methods. The implementation of Page Objects keeps the references to the DOM separated from the test logics, exposing to the tests only the methods that encapsulate any structural details. This grants code reuse, reinforces test robustness, and reduces the test maintenance effort by delimiting the structural breakages in the Page Objects only. An example of a Page Object applied to the test script reported in Figure 1 is shown in Figure 4: in this example, the test script does not refer directly to the structure of the login Web page, but instead uses indirection through the LoginPage Page Object, via the doLogin() method, which encapsulates all the references of the locators needed to retrieve and interact with the Web elements.

Even though there are empirical evidence on the benefits of adopting the Page Object pattern [24], its manual application may require a remarkable effort, since the tester would have to learn in advance about the structure of the Web application under test and implement the Page Objects accordingly. Several attempts to automatize such activities and automatically generate Page Objects during Web crawling have been proposed, like InwertGen [42], APOGEN [37], and ASSESSOR [27]. InwertGen implements an automated incremental test case generation that follows Page Objects generation based on DOM states analysis. Similarly, APOGEN crawls the Web pages and automatically derives a Page Object model based on clustering and static analysis, incorporating some best practices. ASSESSOR extends the functionality of Selenium visual recording to automatically generate Page Objects from recorded scripts via manual annotations. Further on, Biagiola *et al.* [3] formulated a search-based approach to exploit a pre-existent Page Objects model to automatically generate test cases based on feasible paths.

Nevertheless, the automation of the Page Objects generation process can still present some criticalities, such as: (i) *realignment cost* in case of test breakages, since Page Objects still rely on locators, (ii) *methods generation granularity*, sometimes being too fine-grained (e.g., a method for each interaction on a Web element) and sometimes too broad (e.g., some nested functionalities might

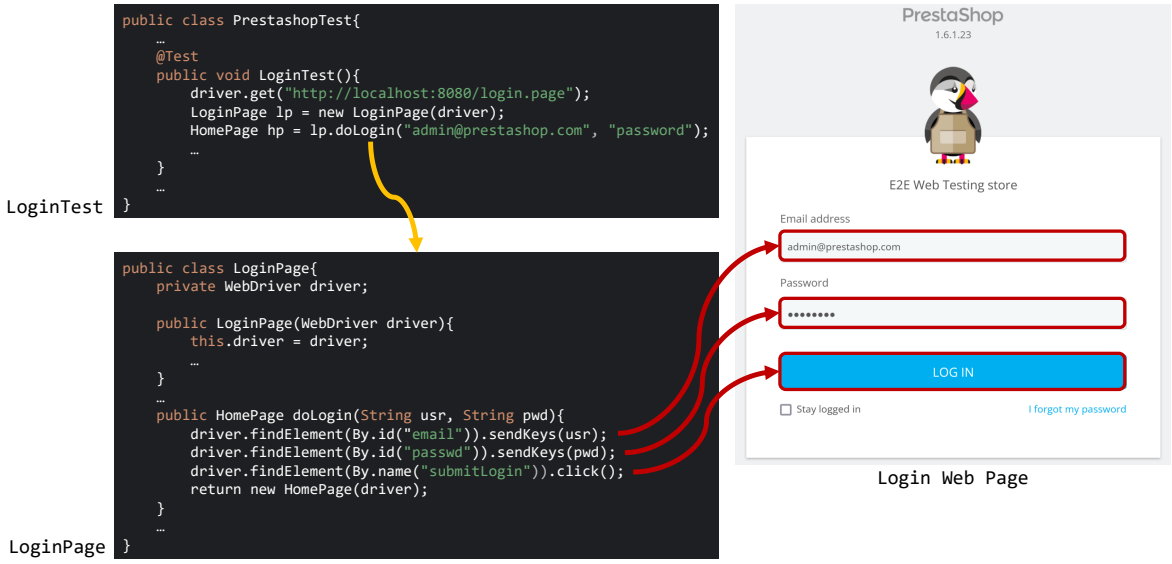

Figure 4: A Selenium login test script with associated Page Object (left) and the corresponding login Web page under test (right).

be missing), and (iii) *Page Objects readability*, which may contain constructs hindering the code comprehension.

2.3 Gamification

Gamification is an innovative paradigm that has emerged in both academic and industrial areas as a trend to integrate gaming elements to non-gaming contexts, in order to improve user engagement and make tedious tasks more appealing [12]. This approach has gained traction even in software engineering, where gamification can support the software engineering process on several levels [10, 35]. In requirements elicitation, it can motivate stakeholders by introducing a voting system tailored for better understanding of requirements, or assigning rewards to participants identifying new requirements. In the context of software development, developers may be rewarded according to the outputs provided by code quality inspection platforms (e.g., SonarQube[3]), or based on their actual contributions to the teamwork. In the testing domain, gamification can lead to higher code coverage and mutation scores, when testing is designed as quests to complete.

At its core, gamification aims to create an engaging and interactive experience, by asking to the user to complete tasks, which provides an overall sense of fulfillment and challenge. One of the key components of gamification is the *customizable profile*, which allows users to personalize their identity. A profile is enriched with rewards obtained from completing tasks. Rewards may include experience points that unlock new levels and titles, synthesizing the user skills and status, as well as icons displayable on their profile, serving as visual trophies. Tasks assigned to users may include *achievements* and *dailies*. Achievements are the primary elements driving the gamification process, motivating users to obtain them. Each achievement is typically featured by a recognizable name, a clear descriptive goal, a graphical icon, and a reward provided once completed. Achievements require users to perform potentially lengthy and difficult tasks in exchange for experience points and unique icons. Given that unpredictability is a core drive in gamification frameworks [6], some achievements may be classified as *Easter eggs* – i.e., hidden features embedded for users to discover after the completion of specific actions, with the intention to surprise – making them extremely challenging or quirky to obtain. Dailies are time-limited achievements randomly assigned to users, typically less challenging, which further enhance unpredictability and keep users motivated even after completing all other achievements.

One of the most prominent gamification frameworks adaptable to testing is OCTALYSIS [6]. The framework emphasizes core drives that motivate users, encompassing concepts such as enjoyment, socialization, and sense of control. These drives are formulated balancing between positive and negative motivators, to provide rewards and satisfaction in accomplishing tasks and to make the user to feel fearful yet intrigued to learn more and become more skilled. Furthermore, the framework works on motivators that stimulate the user intrinsic (e.g., creativity in completing tasks, freedom to explore) and extrinsic (e.g., planning goals, following instructions) perspectives [19].

A plethora of works in educational field have combined testing activities with gamification [4] to develop better testing habits. One of the first proposal was designed by Bell *et al.* [2], disguising testing concepts into stories and quests integrated in a MMORPG-like (Massively Multiplayer Online Role Playing Game) environment. Other studies covered topics as mutation testing, exploratory testing, test driven development, code coverage, and continuous integration [16, 31, 36], showing how testers involved in a gamification experience are generally more engaged and achieve better results. Straubinger and Fraser [39] identify gamification as the most effective practice for influencing the developer behavior towards testing, when applied directly in the working environment. To this end, they implement a plugin for IntelliJ to rewards with achievements the

[3]https://www.sonarsource.com/products/sonarqube/

developers following positive testing practices (e.g., set breakpoints and add assertions to passed tests).

According to the literature, gamification applied to software engineering mostly focuses on unit testing [10, 17]. Nevertheless, a framework to employ gamification to GUI testing was proposed and validated by Cacciotto *et al.* [9]; in this work, the authors formulate metrics to evaluate test performance, such as page coverage, and introduce gamification aspects like leaderboards and Easter eggs. Another proposal is GERRY [20], a Google Chrome extension that enhances Capture & Replay GUI testing with gamification elements and reporting. Like in the work by Straubinger and Fraser [39], an IntelliJ plugin named GIPGUT [19] was proposed to influence test effectiveness, by implementing achievements, dailies, and user progression designed for Selenium WebDriver test implementation and execution. This later work is more discussed in Section 4.3 as we intend to integrate it with TestQuest.

3 TestQuest Tool

TestQuest is an ongoing development project we are currently implementing as a plugin for IntelliJ IDEA, to support locators robustness with an integrated gamification infrastructure. The process describing how TestQuest tool works is sketched in Figure 5. During testing and maintenance, the tester can create and edit the test artifacts, implementing locators, encouraged to perform actions aimed at completing tasks (dailies and achievements) based on the locators robustness heuristics and best practices found in literature. For example, since absolute XPaths are known to be fragile, a daily task might ask the tester to convert a subset of absolute XPaths in a test suite into relative XPaths. TestQuest monitors the changes over the test artifacts following a test suite execution, extracting and analyzing locators to detect dailies and achievements completion based on the passed/failed tests. Eventually, the tester profile is updated according to the completed tasks, rewarding them with titles, icons, and experience points that may lead to a level up. The tester profile is also periodically updated to refresh the lists of completed and active tasks, by removing expired dailies and assigning new ones. This process iterates as long as TestQuest is active and the tester keeps making changes over the test artifacts.

TestQuest is currently implemented in Kotlin, with the persistence layer managed via XML files, while the target test artifacts are expected to be developed in Java with Selenium WebDriver libraries.

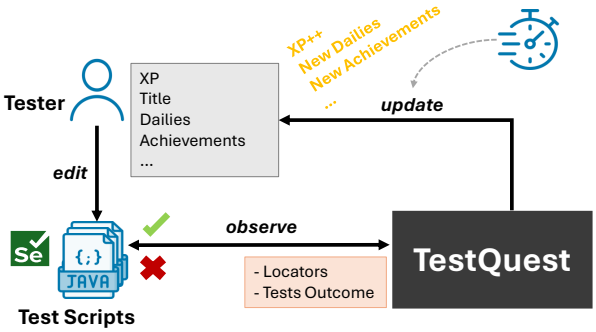

Figure 5: The TestQuest approach.

3.1 TestQuest Architecture

The internal architecture of TestQuest, including the needed external libraries, is shown in the UML component diagram of Figure 6.

The *Test Monitor* orchestrates the whole gamification process shown in Figure 5, working as a listener for any event occurring on the test artifacts following a test suite execution in Selenium; events of interest are additions, removals, and modifications of locators which resulted in passed or failed tests. The *Locators Extractor* parses the test artifacts to retrieve all the locator data (i.e., locator types, locator values, and references to the classes and methods they are part of), once events of interest are detected. The *Gamification Manager* implements the gamification infrastructure, modeling achievements and dailies, and managing the tester progressions. It performs analysis over the extracted locators, based on tests outcome, to evaluate whether some tasks have been completed and, if so, provides feedback to the tester and updates their profile accordingly, removing any daily completed, and assigning rewards (e.g., experience points). The persistence layer of the tester profile is managed via XML files reflecting their progressions. Finally, the *GUI Manager* is responsible for updating the GUI of TestQuest immediately after any tester progress has been detected.

3.2 TestQuest Gamification Infrastructure

According to the literature introduced in Section 2.1, we identified five best practices to support locators robustness. [13, 22, 23, 25, 28, 30, 33]. The majority of the following guidelines pertain to XPaths, as they are known to be the most versatile locator strategy [25]:

(1) Prioritize *id*, *XPath*, and *linkText* locator types, as they have been empirically identified as the most robust, flexible, and easier to use and repair strategies;
(2) Keep locator values readable and short, to improve test cases understandability;
(3) Implement XPath locators which contain as few predicates and levels as possible, since they are more likely to survive DOM evolution;
(4) Prioritize XPath locators with predicates about *id*, *name*, *class*, *title*, *alt*, and *value* properties, that are more likely unchanged in software evolution;
(5) Avoid XPath locators with predicates about GUI structure, internal app structure, or implementation choices (e.g., *width*, *src*, or *onclick* including Javascript code), as they refer to details which are likely to change.

Based on the aforementioned guidelines, three main aspects characterize the gamification infrastructure of TestQuest: *Achievements*, *Dailies*, and *Progression*, in the following described.

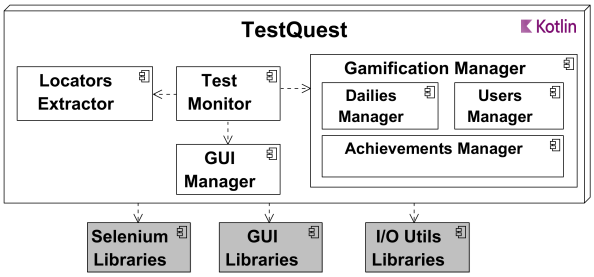

Figure 6: The TestQuest architecture.

3.2.1 Achievements. Each achievement in TestQuest is characterized by a catchy name to make it easily distinguishable, a description clarifying the achievement goal required to complete it, and a reward as experience points and unlockable content (i.e., title or graphical icon) showcased by the user achieving it.

It is expected that achievements are complex tasks, therefore some of them may require long test sessions before completion, while others can provide assistance and reward the beginners (e.g., implement the first locator). Along with achievements that straightly target locators robustness, adhering to the aforementioned guidelines, we have designed other achievements that we speculate as non-damaging to test robustness, yet challenging and enjoyable, including Easter eggs requiring the implementation of locators using creativity (e.g., use every locator type at least once) and bad practices to turn into lessons learned by the tester (e.g., implementing a weak locator may result in a notification making the tester aware of that). This approach can contribute to improve the test quality by capturing the tester engagement [6].

The full list of achievements is made accessible to the user, but some of them are only partially exposed, in particular Easter eggs which provide only their name, challenging the user in achieving them through experimentation.

As an example, achievements that may both challenge the tester and lead to test robustness improvement are shown in Figure 7. The achievements in the first column (**I did it!**, **Keep it short**, **Perfectionism**) are classic achievements that have robustness as clear goal: **I did it!** achievement may reward newbie tester, trying to engage them from the very beginning, whereas **Perfectionism** achievement is more challenging and thus grants the tester with a title reward and higher experience points; this achievement involves the computation of fragility metrics [13, 28] to precisely compare the robustness between locators.[4] On the other hand, the achievements in the second column (**Strength lies in differences**, **The Clone War**, **Ops... I made a mistake!**) all challenge the tester with bizarre requests, that could still lead to locators robustness and actual experience gained, and teach them even from failing (e.g., worsening a locator robustness by making it longer, like changing "//div[1]//input[1]" into "/html/body/div[1]/span/.../input[1]", see Figure 3). Notice that TestQuest assists the tester by reverting the actions that led to a bad practice identified by some achievements (e.g., once a locator robustness is worsened or if a locator crashes after a modification, the change is undone), as well as trying to detect any cheating behavior (e.g., slightly changing the same locator multiple times only to obtain an achievement).

3.2.2 Dailies. Dailies are time-based tasks assigned to the user in exchange of some small reward. Differently from achievements, they are easier to achieve and can be repeated in time, thus sustaining the user engagement even when no more achievements have to be completed. In TestQuest, a daily is characterized by a name, a description about the task to perform, and a reward in the form of experience points assigned to the tester who completes it, based on its complexity. Up to 3 dailies are randomly assigned to each tester, expiring and regenerating every 24 hours, being discardable by the tester once per day in exchange of another random daily. Examples of dailies could be simplified variations of the achievements shown in Figure 7, such as a **Keep it short** daily task which asks the tester to reduce the level of one XPath locator only. Notice that some assigned dailies may not be feasible if locators are already optimized, such as the aforementioned daily when the implemented locators are all *id*-based; we provided the discard feature of dailies in order to prevent such issue.

[4]A Fragility Coefficient can estimate the robustness of an XPath expression on a [0-1] scale; for instance, "//*" returns 0 as fragility score, since it represents the shortest and most robust value for an XPath expression, while an expression containing all levels with predicates from root to target element returns 1, see "/html/body/div[1]/span/.../input[1]" from Figure 3.

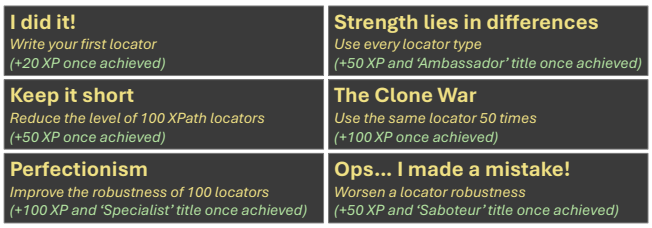

Figure 7: TestQuest achievements.

3.2.3 Progression. In TestQuest, the users are supported with a dedicated GUI window showing their progression, organized into two switchable panels about user profile and achievements. The user profile panel includes a username, a title reflecting the user status in a classic role-playing game fashion, a level as a number synthesizing the user skills, with a progression bar exposing experience points needed to reach the next level, a current profile picture (with edit option to select a new one from those already obtained), a list of assigned dailies, and a list of completed achievements; the user panel is added to enforce the sense of ownership and in encouraging completing more tasks [19], as it is periodically updated based on user progressions. The achievements panel lists the ongoing achievements with the associated progressions. Every user starts at level 1; a level-up mechanism is set from level 1 to level 20 (the maximum level at the moment), based on experience points that scale with each level, requiring more experience points as the user progresses through the levels. Figure 8 shows a mockup of TestQuest in action. In the Figure, the **Fix it!** achievement challenges the user to convert a fragile absolute XPath into a more robust relative XPath. Upon completion, after the test suite execution, the user is rewarded with *10* experience points and an achievement icon, advancing them to the next level and granting a new title (i.e., from *Newbie* to *Novice*).

4 Limitations and advancements

In the following, we discuss the current limitations of TestQuest and the advancements we are considering to improve the tool.

4.1 Multiplayer Experience

We are considering concurrent multiplayer management, as the plugin currently operates at the single-player level only. We aim at introducing features to inspect other users' profiles and general leaderboards to display the global effects on achievements (e.g., display achievements rarity measured by the percentage of users who have obtained them), as well as adding special achievements involving challenges against users.

4.2 Page Objects Gamification

Since the Page Object is a consolidated pattern for End-to-End Web testing [15, 25], as it both helps reinforcing tests robustness and supporting test maintenance, we aim at adapting the Page Object concepts into gamification and integrate them within TestQuest, according to the following best practices and guidelines we have synthesized from the literature [15, 25, 34, 37, 40] presented in Section 2.2:

(1) Page Objects should focus on significant Web elements and pages only (e.g., a button that only changes the background color of a Web page might be neglected from Page Object implementation);
(2) Page Objects should never expose the internal details of a Web page (e.g., locators should only be referenced within Page Objects methods, see Figure 4);
(3) Page Objects should never manage the logic of a test (e.g., assertions should only be formulated within test methods);
(4) Different results for the same action should be modeled as different Page Object methods (e.g., successful and failed login);
(5) Page Objects might share common functionalities, introduced as methods of ancestors within the Page Objects model (e.g., a "back to home" functionality might be translated into a method offered by a NavigationBar Page Object);
(6) Page Object methods might return other Page Objects to model the user's exploration through the application under test (e.g., successfully executing a doLogin() method from LoginPage Page Object might return a HomePage Page Object, see Figure 4).

Inspired by the aforementioned best practices, achievements and dailies should lead the tester in developing test artifacts and Page Objects of high quality, by keeping high their engagement. For instance, a tester might be asked to encapsulate any structural reference to the DOM within the Page Objects, addressing point **(2)** of the aforementioned guidelines, or to implement an ancestor Page Object in case of shared functionalities, addressing point **(5)**, while the gamification infrastructure may check whether no locators are referred within test logics and no duplicated methods are defined in the Page Objects model, rewarding the implementations that adhere to these exemplified cases.

4.3 Integration with GIPGUT Plugin

We aim at integrating TestQuest in Gamification Intellij Plugin for GUi Testing (GIPGUT) [19], an IntelliJ IDEA gamification plugin to enhance GUI testing with Selenium WebDriver. The plugin implements achievements and dailies, as well as a user progression mechanism, tailored around actions over the GUI performed with Selenium tool, such as interactions over Web elements or exploration of specific Web pages. The gamification infrastructure implemented in the tool adheres to the OCTALYSIS framework [6], designing core drives as factors motivating the user engagement in gamification aspects (e.g., the *Ownership* core drive motivating the user in obtaining rare unlockable contents). The plugin is currently implemented in Kotlin and is composed of two modules to manage both static and dynamic aspects of the gamification and test infrastructures, respectively, where Selenium events from test executions are collected and returned for gamification analysis (e.g., to count the progress of an achievement about click methods called during test execution). At the moment, the plugin does not support any gamification aspect pertaining test robustness and locators.

5 Conclusions

Web applications have become fundamental to our daily activities, demanding effective testing approaches to keep test artifacts aligned with the rapid software evolution. Various automated testing techniques and frameworks have been proposed to reduce such effort, yet locators – crucial tools to reference widgets within Web pages – are known as the main reason of test breakages, requiring costly and tedious iterations to repair them. In this paper, we have introduced TestQuest, a tool under development which adapts gamification to Web testing, by rewarding the tester engagement and enhance test quality when robust locators are implemented. As future work, we aim at addressing the limitations discussed in Section 4. In particular, we are working on TestQuest to support the outlined Page Object gamification key concepts, as well as introducing multiplayer and advanced persistence aspects, to manage special achievements involving challenges against other users and accessible leaderboards. Further, we plan to integrate TestQuest with GIPGUT [19], a prototype gamification plugin for GUI testing, in order to exploit its capability in capturing GUI events using Selenium. Finally, we intend to conduct evaluation experiments of TestQuest with students and professionals, and to extend the gamification infrastructure considering additional Web testing best practices and patterns found in literature.

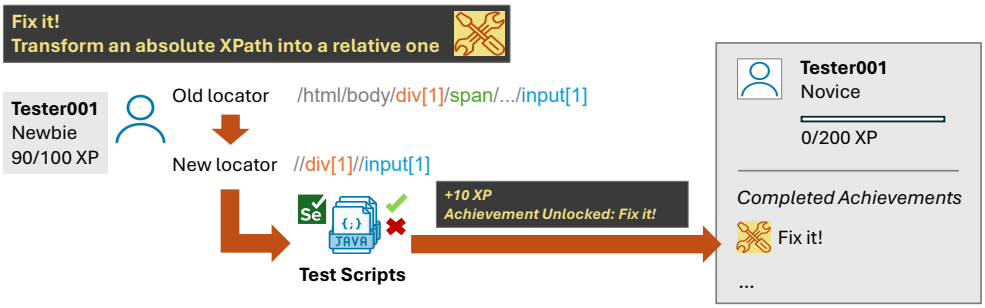

Figure 8: Gamification applied to improve locators robustness.

Acknowledgments

This study was partially carried out within the "EndGame - Improving End-to-End Testing of Web and Mobile Apps through Gamification" project (2022PCCMLF) – Next Generation EU within the PRIN 2022 program (D.D.104 - 02/02/2022 Ministero dell'Università e della Ricerca). This manuscript reflects only the authors' views and opinions and the Ministry cannot be considered responsible for them.

References

[1] Kartik Bajaj, Karthik Pattabiraman, and Ali Mesbah. 2015. LED: Tool for synthesizing Web element locators. In *30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 848–851.

[2] Jonathan Bell, Swapneel Sheth, and Gail Kaiser. 2011. Secret ninja testing with HALO software engineering. In *Proceedings of the 4th International Workshop on Social Software Engineering (SSE)*. 43–47.

[3] Matteo Biagiola, Filippo Ricca, and Paolo Tonella. 2017. Search based path and input data generation for Web application testing. In *9th International Symposium on Search-Based Software Engineering (SSBSE)*. Springer, 18–32.

[4] Raquel Blanco, Manuel Trinidad, Maria Jose Suarez-Cabal, Alejandro Calderón, Mercedes Ruiz, and Javier Tuya. 2023. Can gamification help in software testing education? Findings from an empirical study. *Journal of Systems and Software (JSS)* 200 (2023), 111647.

[5] Shaoheng Cao, Minxue Pan, Yu Pei, Wenhua Yang, Tian Zhang, Linzhang Wang, and Xuandong Li. 2024. Comprehensive Semantic Repair of Obsolete GUI Test Scripts for Mobile Applications. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering (ICSE)*. 1–13.

[6] Y.K. Chou. 2015. *Actionable Gamification: Beyond Points, Badges, and Leaderboards*. Createspace Independent Publishing Platform.

[7] Shauvik Roy Choudhary, Dan Zhao, Husayn Versee, and Alessandro Orso. 2011. WATER: Web application test repair. In *Proceedings of the First International Workshop on End-to-End Test Script Engineering (ETSE)*. 24–29.

[8] Diego Clerissi, Giovanni Denaro, Marco Mobilio, and Leonardo Mariani. 2024. Guess the State: Exploiting Determinism to Improve GUI Exploration Efficiency. *IEEE Transactions on Software Engineering (TSE)* 01 (2024), 1–18.

[9] Riccardo Coppola, Tommaso Fulcini, Luca Ardito, Marco Torchiano, and Emil Alègroth. 2023. On Effectiveness and Efficiency of Gamified Exploratory GUI Testing. *IEEE Transactions on Software Engineering (TSE)* (2023).

[10] Gabriela Martins de Jesus, Fabiano Cutigi Ferrari, Daniel de Paula Porto, and Sandra Camargo Pinto Ferraz Fabbri. 2018. Gamification in software testing: A characterization study. In *Proceedings of the III Brazilian Symposium on Systematic and Automated Software Testing (SAST)*. 39–48.

[11] Marco De Luca, Anna Rita Fasolino, and Porfirio Tramontana. 2024. Investigating the robustness of locators in template-based Web application testing using a GUI change classification model. *Journal of Systems and Software (JSS)* 210 (2024), 111932.

[12] Sebastian Deterding, Miguel Sicart, Lennart Nacke, Kenton O'Hara, and Dan Dixon. 2011. Gamification: Using game-design elements in non-gaming contexts. In *CHI'11 Extended Abstracts on Human Factors in Computing Systems*. 2425–2428.

[13] Sergio Di Meglio and Luigi Libero Lucio Starace. 2024. Towards Predicting Fragility in End-to-End Web Tests. In *Proceedings of the 28th International Conference on Evaluation and Assessment in Software Engineering (EASE)*. 387–392.

[14] Anna Rita Fasolino and Porfirio Tramontana. 2022. Towards the generation of robust E2E test cases in template-based Web applications. In *48th Euromicro Conf. on Software Engineering and Advanced Applications (SEAA)*. IEEE, 104–111.

[15] Martin Fowler. 2013. *Page Object*. https://martinfowler.com/bliki/PageObject.html

[16] Gordon Fraser, Alessio Gambi, Marvin Kreis, and José Miguel Rojas. 2019. Gamifying a software testing course with code defenders. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education (SIGCSE TS)*. 571–577.

[17] Tommaso Fulcini, Riccardo Coppola, Luca Ardito, and Marco Torchiano. 2023. A review on tools, mechanics, benefits, and challenges of gamified software testing. *Comput. Surveys* 55, 14s (2023), 1–37.

[18] Tommaso Fulcini, Giacomo Garaccione, Riccardo Coppola, Luca Ardito, and Marco Torchiano. 2022. Guidelines for GUI testing maintenance: A linter for test smell detection. In *Proceedings of the 13th International Workshop on Automating Test Case Design, Selection and Evaluation (A-TEST)*. 17–24.

[19] Giacomo Garaccione, Tommaso Fulcini, Paolo Stefanut Bodnarescul, Riccardo Coppola, and Luca Ardito. 2024. Gamified GUI testing with Selenium in the IntelliJ IDE: A Prototype Plugin. *ArXiv* abs/2403.09842 (2024). https://api.semanticscholar.org/CorpusID:268510502

[20] Giacomo Garaccione, Tommaso Fulcini, and Marco Torchiano. 2022. GERRY: A gamified browser tool for GUI testing. In *Proceedings of the 1st International Workshop on Gamification of Software Development, Verification, and Validation (GAMIFY)*. 2–9.

[21] Mouna Hammoudi, Gregg Rothermel, and Andrea Stocco. 2016. WATERFALL: An incremental approach for repairing record-replay tests of Web applications. In *Proceedings of the 24th International Symposium on Foundations of Software Engineering (FSE)*. IEEE, 751–762.

[22] Mouna Hammoudi, Gregg Rothermel, and Paolo Tonella. 2016. Why do record/replay tests of Web applications break?. In *2016 IEEE International Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 180–190.

[23] Hiroyuki Kirinuki, Haruto Tanno, and Katsuyuki Natsukawa. 2019. COLOR: Correct Locator Recommender for Broken Test Scripts using Various Clues in Web Application. In *Proceedings of the 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 310–320.

[24] Maurizio Leotta, Matteo Biagiola, Filippo Ricca, Mariano Ceccato, and Paolo Tonella. 2020. A Family of Experiments to Assess the Impact of Page Object Pattern in Web Test Suite Development. In *Proceedings of 13th IEEE International Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 263–273. https://doi.org/10.1109/ICST46399.2020.00035

[25] Maurizio Leotta, Diego Clerissi, Filippo Ricca, and Cristiano Spadaro. 2013. Comparing the Maintainability of Selenium WebDriver Test Suites Employing Different Locators: A Case Study. In *Proceedings of 1st International Workshop on Joining AcadeMiA and Industry Contributions to testing Automation (JAMAICA)* (Lugano, Switzerland). ACM, 53–58. https://doi.org/10.1145/2489280.2489284

[26] Maurizio Leotta, Diego Clerissi, Filippo Ricca, and Paolo Tonella. 2013. Capture-Replay vs. Programmable Web Testing: An Empirical Assessment during Test Case Evolution. In *Proceedings of 20th Working Conference on Reverse Engineering (WCRE)* (Koblenz, Germany). IEEE, 272–281. https://doi.org/10.1109/WCRE.2013.6671302

[27] Maurizio Leotta, Antonio Molinari, and Filippo Ricca. 2022. ASSESSOR: a PO-Based WebDriver Test Suites Generator from Selenium IDE Recordings. In *Proceedings of 15th International Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 389–399. https://doi.org/10.1109/ICST53961.2022.00045

[28] Maurizio Leotta, Filippo Ricca, and Paolo Tonella. 2021. SIDEREAL: Statistical Adaptive Generation of Robust Locators for Web Testing. *Journal of Software: Testing, Verification and Reliability (STVR)* 31 (2021). Issue 3. https://doi.org/10.1002/stvr.1767

[29] Maurizio Leotta, Andrea Stocco, Filippo Ricca, and Paolo Tonella. 2015. Using Multi-Locators to Increase the Robustness of Web Test Cases. In *Proceedings of 8th IEEE International Conference on Software Testing, Verification and Validation (ICST)* (Graz, Austria). IEEE, 1–10. https://doi.org/10.1109/ICST.2015.7102611

[30] Maurizio Leotta, Andrea Stocco, Filippo Ricca, and Paolo Tonella. 2016. ROBULA+: An Algorithm for Generating Robust XPath Locators for Web Testing. *Journal of Software: Evolution and Process (JSEP)* 28, 3 (2016), 177–204. https://doi.org/10.1002/smr.1771

[31] Olivier Liechti, Jacques Pasquier, and Rodney Reis. 2017. 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, 9–15.

[32] Michel Nass, Emil Alégroth, Robert Feldt, Maurizio Leotta, and Filippo Ricca. 2023. Similarity-based Web Element Localization for Robust Test Automation. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 32, 3, Article 75 (2023), 30 pages. https://doi.org/10.1145/3571855

[33] Vu Nguyen, Thanh To, and Gia-Han Diep. 2021. Generating and selecting resilient and maintainable locators for Web automated testing. *Software Testing, Verification and Reliability (STVR)* 31, 3 (2021), e1760.

[34] Page Object. 2023. *Page Object Models*. https://www.selenium.dev/documentation/test_practices/encouraged/page_object_models/

[35] Oscar Pedreira, Félix García, Nieves Brisaboa, and Mario Piattini. 2015. Gamification in software engineering - A systematic mapping. *Information and Software Technology* 57 (2015), 157–168.

[36] Wei Ren. 2023. Gamification in test-driven development practice. In *Proceedings of the 2nd International Workshop on Gamification in Software Development, Verification, and Validation (GAMIFY)*. 38–46.

[37] Andrea Stocco, Maurizio Leotta, Filippo Ricca, and Paolo Tonella. 2017. APOGEN: Automatic Page Object Generator for Web Testing. *Software Quality Journal (SQJ)* 25, 3 (2017), 1007–1039. https://doi.org/10.1007/s11219-016-9331-9

[38] Andrea Stocco, Rahulkrishna Yandrapally, and Ali Mesbah. 2018. Visual Web test repair. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*. 503–514.

[39] Philipp Straubinger and Gordon Fraser. 2024. Improving Testing Behavior by Gamifying IntelliJ. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering (ICSE)*. 1–13.

[40] Arie Van Deursen. 2015. Testing Web applications with state objects. *Commun. ACM* 58, 8 (2015), 36–43.

[41] Selenium WebDriver. 2024. *Selenium WebDriver Documentation*. https://www.selenium.dev/documentation/webdriver

[42] Bing Yu, Lei Ma, and Cheng Zhang. 2015. Incremental Web application testing using Page Object. In *2015 Third IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb)*. IEEE, 1–6.