POLITECNICO DI TORINO

Master's Degree in Computer Engineering

Master's Degree Thesis

Development of integrated tools for the gamification of scripted GUI testing practices

Supervisors:

Prof. Riccardo Coppola

Dr. Tommaso Fulcini

Candidate:

Paolo Stefanut Bodnarescul

Academic Year 2022/2023
Torino

Summary

Software testing is a crucial phase of the software development cycle as it identifies defects in the written code, allowing developers to address them before the product is delivered. Despite its importance, this activity is often neglected because it is perceived as expensive, both in terms of money and time.

This work focuses on Graphical User Interface (GUI) testing, whose purpose is to verify that the user interface of the System Under Test (SUT) works as expected. GUI testing is frequently perceived as a monotonous and repetitive activity, often subject to errors, and the absence of immediate feedback on the quality of tests produced discourages developers from executing this kind of testing. Nowadays, mobile and web applications are widespread, so it is essential to assess the usability and functionalities of the interfaces the users interact with.

Many attempts have been made to make software testing activities more enjoyable and stimulating. Gamification, defined as "the use of game design elements in non-game contexts", is one of the most frequently adopted. There are many examples of applying gamification elements to software testing in literature: one of the most notable is Code Defenders, used to teach students mutation testing. The application of gamification elements to GUI testing, particularly in scripted techniques, is limited. This work addressed this gap by introducing gamification elements to scripted GUI testing.

The implementation involved developing a plugin for a popular Integrated Development Environment (IDE), IntelliJ IDEA, and utilizing Selenium WebDriver as the scripted GUI testing framework. Gamification elements were successfully introduced through the tracking of the Selenium WebDriver. However, the plugin alone could not accomplish this. Therefore, the development of an additional library in Java was necessary, initialized by the user before tests to send the data to the plugin. The incorporated gamification elements include progression, user customization, unlockable content, achievements, daily tasks, and an achievement showcase.

After the development, an experimental phase with a small sample of participants, who graduated in computer engineering, was conducted. They had to perform three tasks on a website while using the developed plugin and library. Survey results

demonstrated user enthusiasm for the gamification elements, and an improvement in overall satisfaction was noticed. An increase in their testing efficiency with the introduction of gamification techniques could not be confirmed in the conducted session.

The experiment underscored how the gamification elements improved user satisfaction and the user experience, affirming the tool's potential for future applications and its effectiveness in stimulating developers to engage more in GUI testing.

Acknowledgements

This has been a long and exciting journey that has now reached its end after many ups and downs.
I would like to thank deeply my family, friends, and cats for their support and encouragement throughout my university journey.
Without them, I probably would not be here writing and finishing this work. Their love and understanding have consistently fueled and motivated me during the whole journey.

Thank you.

Table of Contents

List of Tables

List of Figures

Acronyms

API
Application Programming Interface

DOM
Document Object Model

GUI
Graphical User Interface

IDE
Integrated Development Environment

JSON
JavaScript Object Notation

PSI
Program Structure Interface

REST
Representational State Transfer

SDK
Software Development Kit

SUT
Software Under Test

UML
Unified Modeling Language

Chapter 1

Introduction

Software testing is a crucial and indispensable phase of software development because it allows the detection of problems and issues that may arise during development. If not addressed, these could threaten the quality, performance, and reliability of software applications. Despite the importance of testing, this activity is often neglected because it is perceived as expensive, both in terms of money and time. It does not provide tangible and immediate results to the companies, so they often prefer to focus their resources solely on development. Additionally, testing is seen by the developers as a boring and repetitive activity that does not stimulate them during their work.

Testing inadequately or in a very superficial way exposes the software to future risks. Security problems, disservices, and unwanted behaviors may spoil the experience the users have with the software. After identifying the problems, it is necessary to address and fix them. However, the cost, both in terms of money and time, required for post-development is greater than what would have been needed by applying adequate testing during development time. Moreover, releasing software with many bugs ruins the company's reputation, decreasing the user's trust in the company and its future products.

The work of this thesis focuses on Graphical User Interface (GUI) testing, whose purpose is to verify that the user interface works as expected. All the visible elements present on the user interface can be tested and checked in different ways. For instance, the testers can check if a button is present and if it has the correct behavior or if a text area is visible and displays the correct content. GUI testing is a widespread technique, particularly in mobile and web applications to assess the functionality and usability of user interfaces and it is crucial to develop a product the users find enjoyable to use. This type of testing is seen as repetitive, often subject to errors, and the absence of immediate feedback on the quality of the tests produced further alienates the developers from executing these tests.

The goal of this thesis is the prototyping of a solution that would increase the

developers' interest in performing scripted GUI testing. In particular, a plugin for a popular Integrated Development Environment (IDE) has been developed to incentivize developers to perform this type of testing through gamification techniques.

The thesis is structured in the following way:

- Chapter 2 describes the background regarding web application testing, gamification, and gamification applied in software testing.
- Chapter 3 describes the various tools utilized in this thesis and the architecture of the solution developed to reach the thesis goal.
- Chapter 4 describes a testing session conducted on a small sample of users used to validate the plugin's goals and characteristics. Following, there is an analysis of the users' answers to the survey administered after the session.
- Chapter 5 describes the limitations of the developed tool and the improvements that can be applied to the solution for future applications.

Chapter 2

Background

The main focus of this thesis is gamification applied to GUI testing of web applications. For this reason, before examining the state of the art of GUI testing and gamification in literature, it is important to define what is software testing, its advantages and disadvantages, and how it is integrated into software development.

Software testing is the process of validating software to ensure that it meets the initially defined criteria and has no critical defects that could harm the experience of the users utilizing it. Its purpose is to reveal issues and flaws to the developers during the development phase, enabling the possibility to rectify them. This process ensures that the software aligns with the defined requirements and adheres to the technical specifications. The cost of software testing can, as seen in the literature, amount to up to 50% of the total cost of software development [1]. For this reason, software testing is often neglected because it is perceived as an expensive activity that does not produce immediate results. On the contrary, doing adequate testing during the development brings different advantages. The first advantage is economical: doing testing while developing, and not postponing it until after production, is less expensive. The second advantage regards user satisfaction: software with fewer flaws is preferred to one that brings many problems that need time to be fixed. Additionally, it harms the reputation of the company: a user is less prone to choose software products of the same company if they had previously a poor experience. Finally, testing is advantageous in the long term to check if flaws arise in the future or to do a risk assessment. Therefore, testing should be done at all stages of development to avoid various issues and to spare costs that post-development testing may bring.

Software testing can be classified on different parameters. Some of the possible ways of characterizing testing activities are testing *levels*, *phases*, and *methodologies* [2].

Test *levels* take into consideration the size of the component of the Software

Under Test (SUT) that is being tested. The levels start from testing the individual units of source code, called *Unit Testing*, to levels that test the interaction between multiple units (*Integration Testing*), and levels that test the whole SUT as a user would do (*System Testing*). System Testing includes various practices, such as End-to-End testing (E2E), which tests the SUT by executing end user's cases, or GUI testing, which evaluates the visual presentation of the SUT by using its GUI. One representation of the testing levels that is widely used is Cohn's testing pyramid [3] as seen in Figure 2.1.

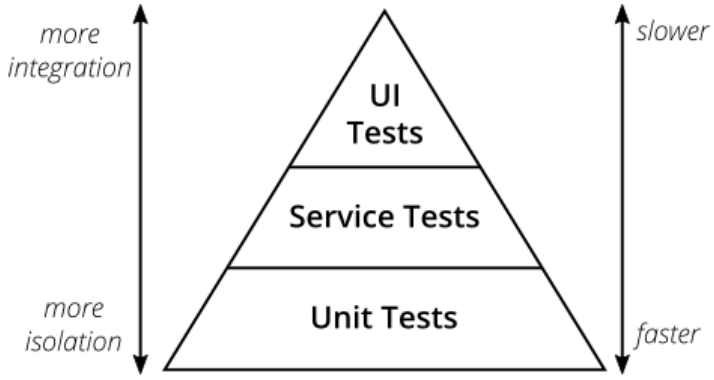

Figure 2.1: The Test Pyramid[1]

The testing can be performed using different *methodologies*. Usually, unit testing and integration testing are performed by utilizing scripted techniques. That means that the developers define programmatically the tests that need to be executed on the SUT [4]. It is also possible to generate test scripts automatically through techniques that fall under the name of *Test Automation*, which is one of the trends in software engineering of the last decades. Many tools are available to generate tests for the developers and their main advantage is the time spent on testing, which is greatly reduced. System Testing is the main level in which these methodologies are used. A brief tool classification [5] can be seen in Table 2.1.

Finally, the *phases* are how the software testing activity is organized. Several frameworks are available, but one of the most popular is the *Software Testing Life Cycle* (STLC) [6]. There are six major phases, as can be seen in Figure 2.2, in every STLC model:

- **Requirement Analysis**: the phase where the test team analyzes the requirements from a testing point of view to identify the testable requirements.

[1] `https://martinfowler.com/articles/practical-test-pyramid.html`

Tool	Description
Manual testing	Manual execution of defined test cases on the SUT
Automation APIs/Frameworks	Manual creation of test scripts that evaluate the GUI of the SUT
Capture & Replay	Manual execution of test sequence used to generate repeatable test scripts
Automated Test Generation Technique	Model of the SUT automatically evaluated by generating sequences of inputs

Table 2.1: Testing Tools

- **Test Planning**: the phase where the methodology, resources, test environment, and test schedule are determined.
- **Test Case Development**: the phase where the tests are implemented and all the test steps defined.
- **Test Environment Setup**: the phase where the test environment defined in the test planning phase is set up.
- **Test Execution**: the phase in which the testers execute the tests based on test cases prepared. This process includes test execution, test maintenance, and bug reporting.
- **Test Cycle Closure**: the phase where the test results are collected and discussed among the team members.

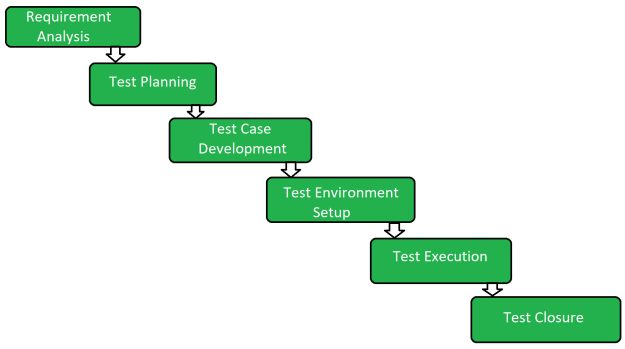

Figure 2.2: Software Testing Life Cycle[2]

[2]https://www.geeksforgeeks.org/software-testing-life-cycle-stlc/

2.1 Web Application Testing

The previous section described a general view of software testing and why it is important in software engineering. Now, the discussion will focus on web application testing and, in particular, on GUI testing of web applications.

Nowadays, web applications have gained immense popularity, leading to the proliferation of various frameworks tailored for their development. They are developed in different languages, for instance, HTML or JavaScript on the client-side and Java, PHP, or Ruby on the server-side. The front-end development focuses on the visual aspect of the web application, the part the user interacts with. Some of the most popular frameworks used for front-end are *React*, *Angular*, and *Vuejs*. Instead, the back-end development focuses on the site's structure, data, and logic. Many frameworks are available also for back-end development like *Laravel*, *Django*, *Spring*, and *ExpressJS*.

The existence of all these variegated and different frameworks makes testing more difficult. The tester must be more careful when dealing with these technologies, especially with the ones that are still in development. Using different frameworks on the front-end and back-end creates variegated web applications: they are heterogeneous. Moreover, they are distributed through a client/server architecture with asynchronous communication (HTTP) to synchronize the application state. These characteristics make web applications more complex to test and the testers are more prone to errors [7].

2.1.1 GUI Testing

The GUI testing deals with the evaluation of the user interface. It is an End-to-End testing technique that verifies the behavior of the SUT through the use of its GUI. Concretely, the tester executes the following operations to ensure the robustness of the GUI:

- Checks the presence of GUI elements
- Ensures that all GUI elements are functional and reachable
- Validates error messages
- Verifies the position of GUI elements

There are mainly two approaches to GUI testing: manual and automated [7].

The manual testing requires the developer to execute the test cases by hand at each iteration. This greatly increases the time the tester must dedicate to test the GUI, making it less efficient. Additionally, it is more prone to errors since it is based entirely on human factors.

The automated testing is based on the automatic execution of test scripts created by the testers at each iteration. There are different ways to create those scripts: they can be developed by using a programming language, or automatically through the use of other tools. Some types of GUI automated testing are:

- **Scripted**: the test scripts are written with automation frameworks of GUI testing and dedicated APIs. An example is *Selenium WebDriver*[3].
- **Capture & Replay (C&R)**: the test sequences are manually registered by interacting directly with the GUI and then they are translated into test scripts automatically. Afterwards, these interactions are emulated at each iteration.
- **Automated Test Generation Technique**: it involves the creation of random or systemic input interactions used to navigate the user interface.
- **Model-based Training**: this technique is based on a model of the GUI. The test cases are automatically generated to cover the paths of the GUI model.

The automated testing needs fewer developers and takes less time. Additionally, it also increases the number of bugs found and the test coverage [8]. The downside is that automated tests are not so flexible and they are often *fragile*: if the GUI of the SUT changes the tests may fail, necessitating developer action to rectify the issues. Moreover, the execution of automated tests is faster but their maintenance and creation require time.

Even though manual testing brings some problems to the table and can be a waste of resources, up to 40% of the development cost, there are some niche cases where it highlights flaws that do not stand out from automated testing. A study on security testing of web applications showed that some issues are identifiable only from careful observations of the testers during manual testing [9]. Problems like authentication, cross-site scripting, and access control were only partially identified by automated testing.

In literature, many references demonstrate and compare the various techniques of web testing. One example is the study that confronts the use of Capture & Replay techniques on the Selenium IDE and the use of scripted testing using Selenium WebDriver [10]. When the scripted approach is used, the cost to develop the tests is greater in terms of the time required. The reason is that developing this kind of test requires a certain level of programming knowledge to create adequate test cases. Scripted testing tries to unify traditional software testing with the web application scenario, by supporting the interaction with the web page and its elements. For

[3] `https://www.selenium.dev/`

instance, a tester can click buttons or fill out some forms programmatically. Even though the time required to develop this kind of testing is higher, the test suite maintenance cost is reduced when using scripted testing. The C&R technique creates *fragile* tests, as mentioned before. A change in the GUI of the SUT makes the old tests fail, requiring the tester to register the new interactions. On the other hand, scripted tests are usually more robust and reliable to changes in the GUI. Therefore, the cumulative cost of scripted web testing becomes lower than the Capture & Replay web testing, with the cost-saving getting amplified on successive releases.

Selenium

A popular framework to perform scripted testing is Selenium WebDriver. It is an open-source tool that allows the automation of web browsers, hence the possibility of testing web applications. With Selenium it is possible to execute different test operations: *acceptance*, *functional*, *performance*, *load*, and *stress*.

Selenium WebDriver permits the simulation of a web browser and it offers many APIs in different programming languages to interact with the GUI of the web application. Hence, the scripted tests are implemented in the preferred programming language and they are usually integrated with *JUnit* to make assertions. The tester can identify the widgets on the web page by using different strategies, offered by Selenium, called *locators*. These strategies are based on the *Document Object Model* (DOM) properties of the web page. For instance, a web element can be obtained by using its id, name, CSS, class name, tag name, or XPath. There is not a locator universally considered as the most robust, it depends on the SUT. It is important to keep in mind that, if the locators are not updated when the SUT evolves, the tests may fail. This is the required test maintenance that was cited before.

Scout

A new typology of automated testing based on Capture & Replay techniques is *Augmented Testing* (AT), which was introduced in Nass et al.'s study [11]. This new method allows the creation of the test suite by interacting with the *Augmented GUI*: an interface that overlaps on the GUI of the original SUT showing additional information.

This technique is based on the use of an *Augmented Layer* that is between the SUT and the tester. Every action (mouse click, dragging...) gets registered by the Augmented Layer and then reproduced on the original GUI. Moreover, some additional metadata are shown on this GUI that help the tester's work. For instance, the elements the user interacted with are highlighted by a colored rectangle around the widget. Therefore, the Augmented Layer receives the information to show by

the SUT, the elaborated metadata and puts them on the same user interface. A diagram of the Augmented Testing workflow can be seen in Figure 2.3.

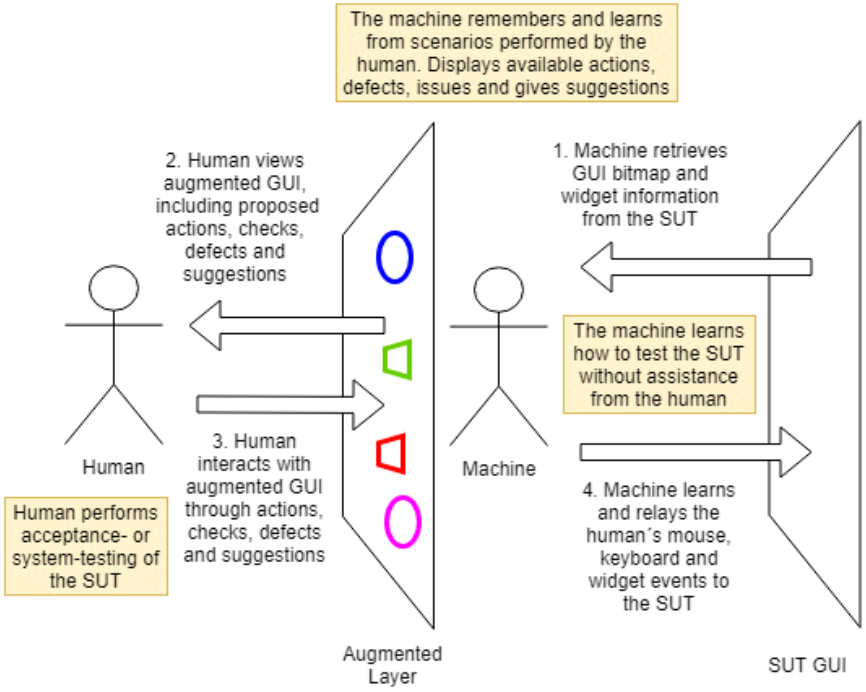

Figure 2.3: Augmented Testing Workflow

An application of Augmented Testing is implemented by *Scout*: a prototype developed in Java that applies the described technique on a web application [12]. Some results from this study showed that creating test cases with Scout is 11% faster than with Selenium. This fact was foreseeable: Capture & Replay techniques are generally quicker to create test cases as explained before, but they are *fragile* and require more time for their maintenance. Additionally, the Scout prototype also brings some limitations. Since the GUI of the SUT shown on Scout is a screenshot taken from the WebDriver approximately every 1 second, it makes some interactions more difficult. For instance, the mouse scroll wheel is disabled, animations are not working, and is computationally heavy.

Finally, to conclude the overview on GUI testing, the main problems that this type of testing introduces will be here listed. Three main issues arise with GUI testing:

- **Flakiness**: a test is called *flaky* when it has both positive and negative results under the same SUT. This could happen for many reasons: there could be

network problems that make the resources unreachable, the execution time is not fixed but varies each time, and the tools are not always completely reliable.

- **Fragmentation**: GUI testing suffers from fragmentation when the result of the test on the same SUT changes based on different environment characteristics. For instance, in hardware fragmentation the SUT is the same but on different devices, in software fragmentation the SUT is the same but on different operative systems and in browser fragmentation the web application is the same but executed on different browsers.

- **Fragility**: a test is *fragile* when it fails if a new version of the SUT is released. This issue has been also cited before, regarding various GUI testing techniques.

2.2 Gamification

Gamification, as defined by Deterding et al., is *"the use of game design elements in non-game contexts"* [13]. Different settings implement elements typically associated with games in contexts whose purpose is not entertainment: an example is *serious games*, which are games whose goal is usually educational. But gamification is different: it is a tool that exploits game elements in non-playful contexts to reach certain objectives. Hence, the main idea is to exploit game design and game mechanics to make activities more enjoyable in environments whose main goal is not entertainment. Incorporating gamification elements offers numerous benefits, such as increasing user motivation and engagement. These elements inspire users to pursue goals by following the established rules, stimulating problem-solving skills, and enhancing social interactions [14].

This technique is fairly new, but in the last decades, it has spread in several areas such as learning, marketing, business, and computer science. In particular, an increase in the use of gamification in software engineering disciplines was reported by Barreto and França [15]. Regarding software testing, the review conducted by Fulcini et al. highlights the fact that gamification is mostly applied to the test creation phase by using simple white-box unit or mutation testing tools, and is mostly used to promote testers' accomplishments and good behavior [2].

In literature, different categorizations of gaming elements in gamified approaches have been proposed, such as the one by Robson et al. [16]. A taxonomy of game elements is provided, categorizing them under game *mechanics*, *dynamics*, and *emotions*. Game mechanics refer to the goal, the rules, and the interactions offered by a gamified system. The game dynamics, instead, concern the interaction between the game and the system during the gamification activities. Finally, the

game emotions refer to the sentiment that the user feels when using the gamified environment. A summary of these concepts can be seen in Figure 2.4.

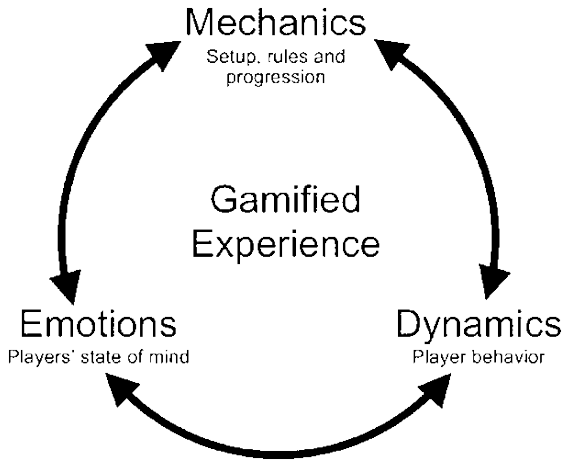

Figure 2.4: MDE (Mechanics, Dynamics, and Emotions) Framework of Gamification Principles

This classification emphasizes the difference between mechanics and dynamics. Game mechanics are common components of the gamified system for all the players, and are independent of the user experience. Instead, game dynamics are variable and are dependent on the user of the gamified activity. However, in most cases, this distinction is not applied in other studies and the two terms are used as synonyms to describe the design of gamification elements in a gamified environment [2].

A systematic approach is needed to build a successful gamified environment. Creating gamified activities that suit the business goal and satisfy the users is a complex task that requires careful planning to be effective. To reach this goal, many frameworks have been proposed that provide a systematic ground for the application of gamified elements. One of the most famous ones and frequently adopted is the *Octalysis* framework, proposed by Chou [17]. As explained by the author, Octalysis emphasizes a *human-focused design*, as opposed to *functional-focused design*. The human-focused design is a design process that optimizes the user's feelings, motivations, and engagement by remembering that users have sentiments and insecurities. On the other hand, functional-focused design's main goal is efficiency: the goal is to get the job done quickly. The Octalysis framework identifies eight core drives that represent human aspects stimulated by gamification as seen in Figure 2.5. These core drives are the reasons that motivate users towards certain activities and that make games fun. Additionally, for each of the eight core drives some popular gamification mechanics are associated as can be seen in the figure.

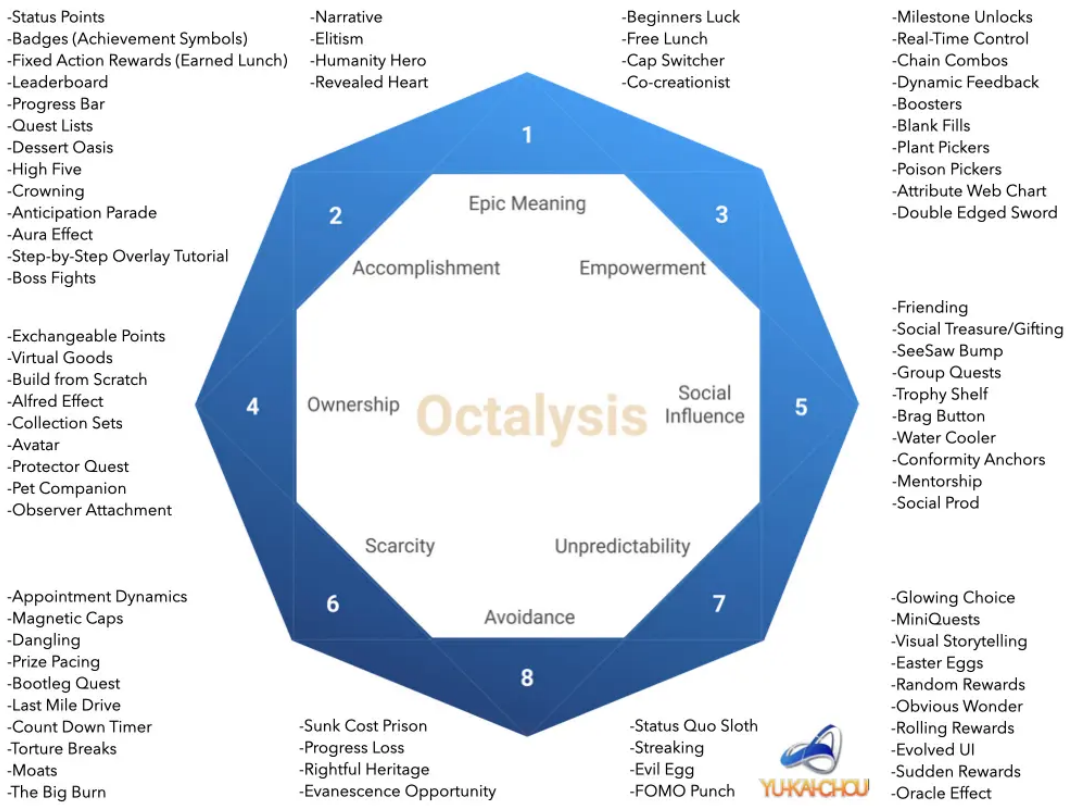

Figure 2.5: The Octalysis Framework

The eight core drives can be summarized as follows [18]:

- **Epic Meaning & Calling**: this is the core drive that makes a user think they were "chosen" to do something, that they have to take part in something greater than themselves. For instance, this comes into play when the user has the "Beginner's Luck" and thinks they are special among all the people.
- **Development & Accomplishment**: this is the core drive of making progress, overcoming challenges, and developing skills. This is the core drive easiest to design gamification elements such as points, badges, and leaderboards.
- **Empowerment of Creativity & Feedback**: this is the core drive where users are engaged in a creative process where they have to try different combinations or they have to figure things out. In this way, users can express their creativity but also receive feedback and respond to it.
- **Ownership & Possession**: this is the core drive where players feel they own something, it being virtual currencies or virtual goods within the system. It

makes the user feel that they want to own better and even more. For instance, spending a lot of time customizing a profile or an avatar makes the user feel more ownership towards it.

- **Social Influence & Relatedness**: this is the core drive that incorporates all social elements that drive people. For instance, companionship, competition, envy, acceptance, or social response. If another player has something the user does not, they are stimulated to reach their level. This drive incorporates also the need of users to draw closer to people, places, or events they can relate. As an example, nostalgia is exploited to make the users buy a certain product.
- **Scarcity & Impatience**: this is the drive that wants people to have something they previously could not have. For instance, rewards that can not be redeemed immediately but after some time.
- **Unpredictability & Curiosity**: this is the drive that stimulates people to go on because they do not know what will happen next. In this way, the brain is engaged and thinks often about what is going to happen. Unfortunately, this is also the drive behind gambling addictions.
- **Loss & Avoidance**: this is the core drive based upon the avoidance of something negative happening. For instance, it could be to avoid losing previous progress or avoid admitting that everything the player did was useless because they decided to quit.

The described eight core drives can also be divided into two higher-level representations: *left-brain* and *right-brain* drives, or *white-hat* and *black-hat* drives.

In Octalysis, the drives on the right part of the octagon are the right-brain drives, meanwhile the drives on the left part are the left-brain drives. Right-brain drives are related to creativity, self-expression, and social aspects. Additionally, they are *intrinsic motivators*: the activity itself is rewarding on its own. There is no need for a reward to be creative or to socialize with others. On the other hand, left-brain drives are related to logic, calculations, and ownership. They are *extrinsic motivators*: the user is motivated because they want to reach a certain goal or obtain a certain reward. This distinction is important because many companies often focus solely on extrinsic motivators. The problem arises when these motivators stop being introduced: user motivation will decrease much lower than before the introduction of gamification. This is the reason why it is important to introduce intrinsic motivators to make the activity fun and rewarding without the addiction of external factors so that the users keep interacting with it.

Regarding the second classification, the drives on the top of the octagon are *positive motivators*, meanwhile the drives on the bottom are considered *negative motivators*. The techniques that use the top drives are called white-hat gamification,

while the ones that use the bottom drives are black-hat gamification. If an activity is engaging because the user is free to express themselves it makes them feel good. If the user is engaged only because they are always in fear of losing something or because they do not know what will happen next, it can sometimes leave an unpleasant feeling. This does not mean that black-hat gamification is bad, it can be also used for productive results. A good gamification system should consider all eight drives to create a positive and productive activity.

To apply the Octalysis framework, the gamification expert must take the octagon and analyze all the gamification elements of the system. For each drive associated with the gamification mechanic, the side of the octagon will retract or expand. If a side crosses the octagon it means that the drive is weak and not exploited, hence the gamification expert should improve that area. If a side is expanded, it means that the drive is strong in the gamification system. Some examples of the framework Octalysis being applied to popular products can be seen on Chou's site [18].

The one described before is only the first level of application of the framework Octalysis, which is enough for the majority of companies to create a better-designed gamified product and experience. However, there are additional levels that dive deeper with more advanced design principles and in-depth analysis. Level two focuses on describing how the gamification elements are distributed among the four phases of a player's journey [Figure 2.6]. Level three goes a step beyond and combines the phases of a player's journey with Bartle's taxonomy of player types [19] [Figure 2.7]. There are in total five levels, but the other ones are not known by the majority of people.

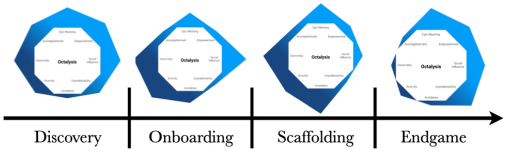

Figure 2.6: Level two Octalysis

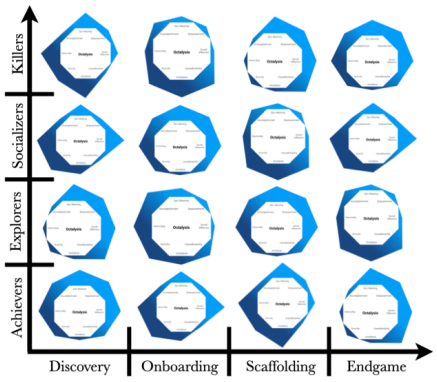

Figure 2.7: Level three Octalysis

Another framework useful to evaluate gamified experiences and the emotional effects of their use on players is GAMEX [20]. This framework identifies several

dimensions that represent the different aspects of gamification: enjoyment, absorption, creative thinking, activation, absence of negative affect, and dominance. It is composed of 27 Likert questions that evaluate how the cited aspects influence the gamified experience. The study conducted by Eppmann et al. demonstrates that GAMEX is a reliable and valid measure, useful both in research and professional contexts, that is easy to apply in different gamification settings.

2.3 Gamification in Software Testing

One of the most expensive phases in software development, both in terms of money and time, is software testing. Even though having an adequate and exhaustive testing suite that reveals as many code flaws as possible is crucial, often software is tested inadequately with the only purpose of respecting some deadlines, and instead most of the time is spent writing code. Additionally, the testing phase is often considered tedious and repetitive by most developers, not able to stimulate the testers' attention.

For these reasons, many attempts have been made to make these activities more enjoyable and more stimulating. Among these, gamification is one of the most frequently adopted [21]. In the study conducted by Martins de Jesus et al., the authors analyzed and classified the use of gamification techniques in software testing according to six perspectives: application context, used gamification elements, gamification goal, testing techniques, and testing process phases. The results showed that the most used gamification elements in software testing are points, leaderboards, and levels, the most used testing level is unit testing and the most addressed technique is functional testing [22].

The purpose of gamification in this context is to improve the quality of the tests and stimulate the developers to do more testing during software development. The final advantage of the integration of gamification elements in software testing is indirectly increasing the software's quality. Below some examples present in the literature are mentioned.

One of the most famous ones is **Code Defenders**, presented by Rojas and Fraser [23]. Code Defenders is a turn-based mutation testing game. The system defines two main roles: the *attackers* whose role is to introduce faults in existing code to make the test suite fail, and the *defenders* whose role is to add tests to the existing test suite to find as many mutants as possible. An example of the defender's user view can be seen in Figure 2.8. The purpose of Code Defenders is educational: engage students with software testing and help them develop good practices.

HALO (Highly Addictive, sociaLly Optimized), as presented by Sheth et al., is an approach to gamifying all the software development phases [24]. It exploits

Figure 2.8: Code Defenders: Defender View

a characteristic element of *MMORPG* (Massive Multiplayer Online Role Play Game): quests. Through quests, daily and tedious tasks are hidden under a playful environment that rewards users with experience points, virtual points, and other prizes. The quests comprehend tasks of various difficulties: some quests are completed in a single task, some quests need the fulfillment of more tasks, and others require collaboration between multiple users. Additionally, quests can be split into sub-quests or must be completed in a particular order.

VU-BugZoo is a digital platform to teach software testing based on a faulty code repository. Its innovation consists in engaging both instructors and students in a bug-hunting experience, where both trophies and weapons play a crucial role, as presented by Silvis-Cividjian et al. [25].

WReSTT-CyLe, acronym of *Web-based Repository of Software Testing Tools Cyber-Enabled Learning Environments*, uses gamification as a strategy to motivate students to learn software testing techniques and improve their knowledge [26].

Clean Game is a tool which applies gamification techniques to *code refactoring* activities [27]. The main purpose of Clean Game is to help in *code smell* identification. It can be used both in academic and corporate environments because the generation of the modules to help identify code smells is based on the input code.

GOAL, proposed by García et al. [28], is a framework that helps the introduction of gamification in every aspect of software development, hence also in software testing. The only type of testing mentioned in this work is unit testing.

Regarding in particular GUI testing, two main tools are found in the literature. A framework that allows building plugins for GUI testing tools exploiting game concepts was defined by Cacciotto et al. [29]. It is based on the Capture & Replay testing technique and a prototype has been developed by Gallotti [30]. The prototype is implemented as a Scout plugin, a tool for Augmented Testing as explained in the previous section. By exploiting the Augmented GUI, the plugin can show additional information to implement gamification elements. For instance, a progress bar is always present at the top of the current page, a star symbol appears if the tester has arrived on a new web page [Figure 2.9], and, at the end of the testing session, a recap of the metrics measured and a score is presented. Additionally, a separate profile page is available, where the user's avatar and their badges are visible.

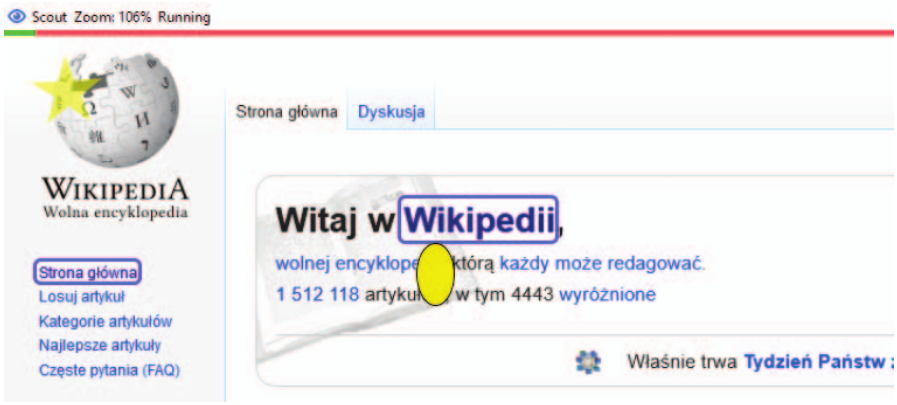

Figure 2.9: Gamified Exploratory GUI Testing Tool

GERRY is another Capture & Replay GUI testing tool which implements an approach based on gamification [31]. The tool has been implemented as a Google Chrome extension which is designed to be compatible with pre-existing GUI testing tools. Two main modes are available: *Interaction Mode* in which the user interacts with the elements available on the web page, and *Reporting Mode* which allows testers to signal issues found during the testing session, such as elements that do not work as intended. The main gamification elements implemented are progress bars, avatars, scores, achievements, leaderboards, and easter eggs.

To end the discussion, it is important also to analyze the limits that a gamified environment brings with it. For instance, in Sasso et al.'s study, the *pointsification* phenomenon is illustrated [32]. This phenomenon explains how the users may try to get as many points as possible, neglecting the tasks that are assigned to them. The testers may prefer to complete easier tasks than more difficult ones, with the only goal of accumulating more points. Or worse, the players may try to cheat the

system to boast how many points they have accumulated in front of other users. These occurrences negate and make useless the introduction of gamification to incentivize the completion of tasks.

It is also important that the gamified environment developed is well thought or it may not bring the expected advantages. Additionally, it should be bug and glitch-free to not compromise the users' experience [33].

Lastly, some gamified environments have limited applicability: they may work well in an academic environment but they may not be well suited for a corporate one. The reasons could be difficulties in integrating the tool into a work environment or the fact that the tool needs a supervisor. In an academic context, the supervisor could be the professor, but in a work environment, there is not a clear indication of who this role should be assigned to [34]. Overall, implementing gamification tools in a corporate context brings some costs, both in terms of money and time, that need to be analyzed carefully to understand if the reported benefits are worth the expense.

Among the cited studies, particularly the literature review on gamification applied to software testing performed by Fulcini et al. [2], there were no presented tools that integrated GUI scripted testing with gamification. Therefore, the work of this thesis goes in this direction, trying to unify these two disciplines.

Chapter 3

Methodology

3.1 Instruments

To reach the goal of implementing gamification elements in a scripted GUI testing environment, the decision was to develop a plugin for an IDE typically used by programmers. Therefore, the first two decisions to be made were selecting the IDE of the plugin and the web application testing framework.

3.1.1 IDE

Selecting the ideal IDE marked one of the first crucial steps in starting the plugin development journey. The choice must take into consideration several aspects that will be briefly analyzed.

The IDE should be of widespread usage, to ensure that users using it will feel a sense of familiarity rather than becoming overwhelmed with all its features. Moreover, a key aspect is its status as a platform where numerous plugins have already been developed and their use is fairly common. These two characteristics confirm the popularity of the IDE but also demonstrate that the development of a plugin is not an unachievable task.

Furthermore, another important aspect to take into consideration is the presence of extensive documentation and the number of examples regarding plugin development. Such resources provide solid help for the developer, serving as a guide through the development process, resolving issues, and enhancing the understanding of the IDE platform and all its capabilities. Hence, the careful choice of the IDE is crucial in ensuring the success and ease of the plugin development but also to ensure facility of use to the end user. In the end, the IDEs taken into account to

develop the plugin for were: *Visual Studio Code*[1], *Eclipse*[2], and an IDE from the *JetBrains*[3] ecosystem.

A comparative analysis will follow up that will highlight the differences between the various available choices.

- Visual Studio Code, or VS Code, is widely known for its lightweight and resource-efficient nature. The minimal system requirements and its minimal interface make it an attractive choice for many developers who prefer a simpler IDE. VS Code also offers an extensive number of available extensions and supports a wide array of programming languages, excelling in web development. The active and dynamic community makes it a good ecosystem to develop a plugin for. On the other hand, there are some issues. It is less preferred than other IDEs like *JetBrains IntelliJ IDEA* for larger-scope projects. Furthermore, its compatibility with languages like Java is less smooth than desired.

- Eclipse stands out for its robust support of Java development. It has an immense ecosystem of plugins available to satisfy all the needs of Java developers. While it is highly customizable and powerful, it introduces a steeper learning curve than some of its counterparts.

- The JetBrains ecosystem is known for top-tier code analysis and refactoring tools. It is very user-friendly and suited for both beginners and advanced developers. The JetBrains ecosystem is also known for all the products it offers, each specializing in specific languages, ensuring versatility. Moreover, the JetBrains marketplace offers a wide array of available plugins and extensions to enhance the developer's productivity. The presence of an active and dynamic community is also a good advantage for plugin development. The existing documentation and the availability of other plugin developers' open-source code is another factor that eases plugin development.

All things considered, the JetBrains platform was chosen for the development of the plugin. Its extensive ecosystem aligns with the project requirements, and it also offers smooth integration with testing frameworks, which is crucial for the plugin's scope.

[1] `https://code.visualstudio.com/`

[2] `https://www.eclipse.org/ide/`

[3] `https://www.jetbrains.com/`

The last decision regards the IDE of the JetBrains ecosystem that the plugin will target. Two options were evaluated: *JetBrains Aqua*[4] and *IntelliJ IDEA*[5].

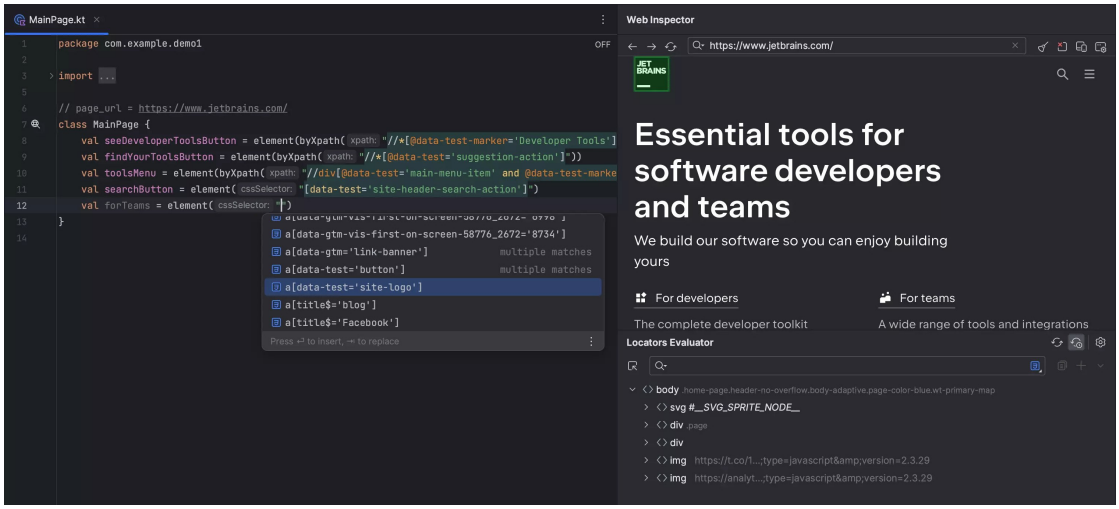

Figure 3.1: JetBrains Aqua Embedded Web Inspector

- JetBrains Aqua is a JetBrains product still in preview mode but it was taken into consideration due to some peculiar characteristics. Aqua is primarily designed for test automation and supports many popular frameworks such as Selenium and Cypress. The presence of a built-in advanced Web Inspector that generates unique CSS and XPath locators for the selected elements of the web page enhances and simplifies the web application testing, as seen in Figure 3.1. Furthermore, Aqua also features a built-in HTTP Client, access to the developer's Docker containers, database support, and a test management system.
- IntelliJ IDEA is the lead JetBrains product for Java and Kotlin development but it also extends support to a wide spectrum of programming languages, thanks to the many available plugins. Its cross-language support, rich ecosystem, and intelligent code assistance make it the IDE that many developers choose worldwide. It is available in two versions: Community and Ultimate Edition, but some premium features are exclusive only to the latter.

The final choice leaned toward developing the plugin for IntelliJ IDEA for several reasons. JetBrains Aqua is still in preview mode and its usage is not yet widespread.

[4] `https://www.jetbrains.com/aqua/`

[5] `https://www.jetbrains.com/idea/`

While the advanced Web Inspector of Aqua was an interesting feature, it is also available in a separate plugin called *Test Automation*[6] for IntelliJ IDEA Ultimate. Moreover, the Test Automation plugin offers rich support for frameworks such as Selenium, Cypress, Selenide, and Playwright among many other features. Hence, the choice of IntelliJ IDEA is the less constrictive one: the developer can use the IDE to which they are accustomed and they have the freedom to exploit the Test Automation plugin if they need it, without giving up the appealing features of JetBrains Aqua. Furthermore, the choice of IntelliJ IDEA is reinforced by one more factor: the test framework used to work with the plugin is Selenium WebDriver, which can be used in Java, making IntelliJ IDEA the ideal target for the plugin.

3.1.2 Web Application Testing Framework

In Test Automation the choice of the right framework is crucial. For the scope of this project which is the testing of the GUI of web applications, three main frameworks have been identified among all: *Selenium WebDriver*[7], *Cypress*[8] and *Appium*[9].

A comparative analysis will follow up to underline the strengths and weaknesses of each approach.

- Selenium WebDriver is an open-source framework widely used for web application testing. It supports multiple testing frameworks and multiple programming languages like Java, C#, Python, and many more. It interacts directly with the browser by simulating user actions such as clicking, navigating, and filling fields. A rich set of APIs is available in Selenium that allows testers to perform all kinds of operations like identifying DOM objects through CSS, XPath, or handling various browser events. Moreover, it also supports cross-browser testing, to execute testing on different browsers like Google Chrome, Safari, or Firefox. On the downside, its setup is not straightforward because it requires setting up the test environment, which can be tedious, and downloading specific browser versions. There are also difficulties working with specific web technologies but despite all of that, it is widely adopted due to its flexibility and extensibility.
- Cypress is relatively recent but has already gained enough traction. It is more beginner-friendly and operates directly in the browser with unique DOM

[6] `https://plugins.jetbrains.com/plugin/20175-test-automation`

[7] `https://www.selenium.dev/`

[8] `https://www.cypress.io/`

[9] `https://appium.io/docs/en/2.1/`

Comparison Factor	Cypress	Selenium	Appium
Primary Use	Automated web testing	Automated web testing	Automated mobile app testing
Language Support	JavaScript	Multiple (Java, C#, Python, etc.)	Java, JavaScript, Python, Ruby, C#
Architecture	Runs directly in browser	Communicates via WebDriver API	Communicates via WebDriver protocol
Test Execution Speed	Fast	Moderate	Moderate
DOM Manipulation	Built-in capabilities	Requires explicit commands	Requires explicit commands
Debugging	Powerful built-in support	Dependent on language and IDE	Dependent on language and IDE
Application Type	Web applications	Web applications	Mobile applications (iOS and Android)
Mobile-Specific Features	N/A	N/A	Gestures, device-specific features, etc.
Setup and Configuration	Minimal	Requires web browser drivers	Requires mobile device drivers or emulators/simulators

Figure 3.2: Comparison between Selenium, Cypress and Appium [35]

manipulation techniques. Unlike Selenium there is no need to add explicit or implicit wait commands in scripts but Cypress waits automatically for assertions and commands. At the beginning, it supported only Chrome but now it has extended its functionalities to Edge and Firefox. Cypress is well suited for modern web applications that use JavaScript and only supports JavaScript for creating test cases. Furthermore, it does not allow the driving of multiple drivers at the same time and it does not support multi-tabs.

- Appium is an open-source framework mainly designed for mobile application testing. It allows the development of tests in various programming languages like Java, C#, and Python and supports various testing frameworks. Appium supports the testing of native, hybrid, and mobile web applications offering many features that are exclusive to mobile devices like screen orientation

and gesture interactions. Like Selenium its setup is not as straightforward as Cypress and requires different configurations for different mobile platforms.

As seen in this brief comparison[35, 36, 37], Cypress and Appium excel in specific niches, while Selenium WebDriver remains a more versatile choice for general test automation. Taking into consideration the choice of the testing framework together with the choice of the target IDE reaffirms that Selenium is the best decision. IntelliJ IDEA works very well with Java and Selenium WebDriver supports Java among the various programming languages. Moreover, its broad browser support, extensive documentation, and rich set of APIs make it the best solution for the scope of this project.

3.2 Architecture

After choosing IntelliJ IDEA as IDE and Selenium WebDriver as the testing framework, the development started. The first and foremost step that was done was examining IntelliJ Plugin SDK[10] (Software Development Kit) and Selenium WebDriver Documentation[11] to understand how to exploit all their features to reach the proposed goal. The IntelliJ Plugin SDK was vital to learning about plugin development and plugin components. It offered a comprehensive understanding of the IntelliJ architecture and the essential components that form a plugin. Simultaneously, the Selenium WebDriver Documentation was needed to provide crucial insight into understanding how Selenium testing works and how it interacts with the WebDriver calls. It was fundamental in the task of intercepting WebDriver calls to gather statistical data and to create a gamified environment through the plugin and its gamification elements.

3.2.1 General Architecture

The components of the project include the development of a plugin for IntelliJ IDEA and the development of an additional library that the user must use together with their Selenium WebDriver.

While the development of the plugin was in progress, it became clear that collecting data from the WebDriver used in the user application during testing was not feasible by only relying on the IntelliJ Platform. The reason is that the IntelliJ Platform offers interfaces and APIs that are mostly functional to code static analysis and not during program execution. As explained later, some solutions

[10] `https://plugins.jetbrains.com/docs/intellij/welcome.html`

[11] `https://www.selenium.dev/documentation/`

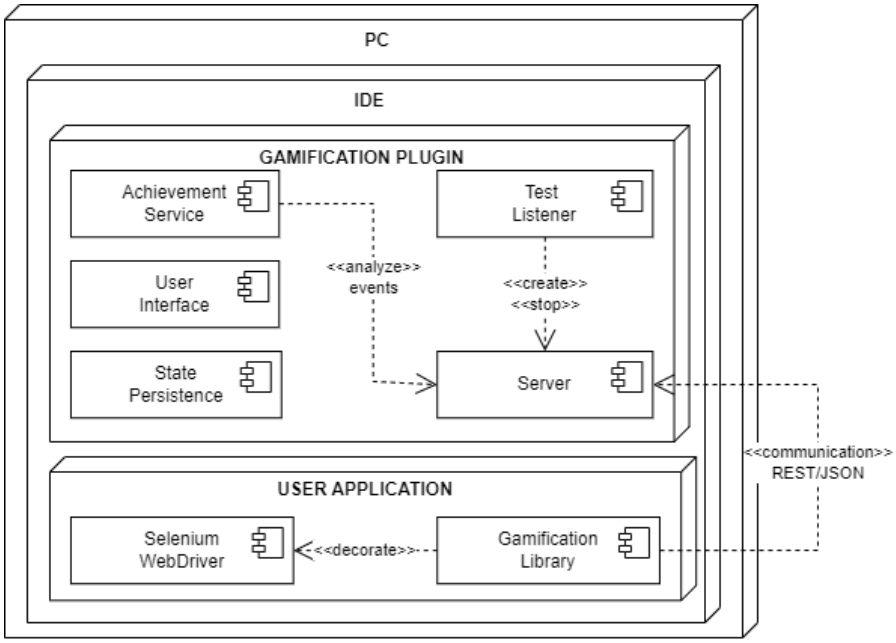

Figure 3.3: UML Deployment Diagram

were examined but they all were hard to implement and easily prone to errors because they needed to alter the user's build environment. For these reasons, the best solution was to develop an external library that would supply the gamification plugin with the data to correctly implement the gamification elements.

The developed library exploits a listener that is embedded in the Selenium framework which allows the execution of tasks before and after each WebDriver call. In this way, the listener collects the data and, at the end of the test, sends it to the plugin. To make the plugin and the library communicate with each other the decision was to use JSON and REST APIs. Before the testing commences, the plugin opens a local server that stays running until the end of the test suite. In the meantime, the library sends the information at the end of each test, so that the plugin can elaborate on them and correctly store the statistics useful for implementing the gamification elements.

Additional crucial parts of the plugin that deserve attention and will be the focus later on, are the *Achievement Service*, the user interface, and the *State Persistence*. The Achievement Service's purpose is to analyze the information and events that the library sends to correctly increase the achievements' progress, one of the gamification elements of the plugin. The State Persistence class is needed to store the plugin's information so the user to not lose their data between different sessions. Finally, the user interface is another important component. It emotionally involves the user in their accomplishments and it must be clear, concise, and

cohesive. Additionally, it must follow the general theme of the IDE to not clash too much with the other components.

An UML Deployment Diagram of the entire project can be seen in Figure 3.3. Each aspect will be discussed in more detail in the following sections. The implementation of both the gamification plugin [38], *Game GUI*, and the gamification library [39] can be found on *GitHub*[12].

3.2.2 Gamification Plugin

The development planning began by examining the IntelliJ Plugin SDK documentation and looking for available open-source code of other plugins online. It soon became clear that the official documentation was lackluster: some concepts were not explained in enough detail and most of the provided examples only covered straightforward scenarios. Hence, a critical step in the development was taking a look at how other developers created their plugins, assuming that their code was open-source and available. The official documentation also advised this approach. Within the IntelliJ Plugin SDK documentation, when cataloging the possible plugin listeners and extension points (plugin components), there are also listed the plugins that have implemented these features. Moreover, for the plugins with available source code, the documentation redirects you to their GitHub page. If that is not enough to solve some doubts, it is also possible to check out IntelliJ IDEA's source code to understand how some features of the IDE were realized directly by the JetBrains team. Furthermore, the active and vast community also plays a crucial point in plugin development. A forum[13] is available online, directly on JetBrains' site, where to ask questions about plugin development and where members of the JetBrains team or other users can answer. In addition to the forum, a *Slack*[14] channel is available which further speeds up communication to solve difficulties and to receive information on new updates.

Programming Language

When the actual development started, the first crucial thing to decide was which programming language to use. It is mandatory to use a JVM (Java Virtual Machine) language to develop a plugin for an IntelliJ Platform, such as IntelliJ IDEA which is the target of the project. At the moment, it is not possible to develop plugins that target these platforms in non-JVM languages. Hence, the choice of the

[12] `https://github.com/`

[13] `https://intellij-support.jetbrains.com/hc/en-us/community/topics/200366979-IntelliJ-IDEA-Open-API-and-Plugin-Development`

[14] `https://slack.com/`

programming language had to be between Java, Kotlin, or other JVM languages. In the end, it was opted to develop the plugin using Kotlin.

Kotlin is very similar to Java but it presents some additional features and solves some issues that Java has. To make some examples of issues solved, in Kotlin *null references* are controlled by the type system, the arrays are invariant by default, there are no raw types, and there are function types. Furthermore, in addition to Java, Kotlin has *lambda functions*, *coroutines*, *null safety*, and many other features.

On the other hand, Java has some characteristics that are not present in Kotlin. For instance, the *ternary operator* or the *static* keyword are not available in Kotlin but are replaced by similar features like the *if condition* or the *companion object* [40].

In the end, Kotlin was chosen as the programming language for the plugin because of the appeal of its additional features. Another fact not to overlook is that Kotlin is less verbose, which means less code. So, overall, it is less prone to bugs.

As mentioned before, Java and Kotlin are very similar hence developing a plugin in Kotlin is not too different than developing one in Java. Existing developers who have their plugin written in Java can easily convert their Java classes to the Kotlin counterpart by using the J2K (Java to Kotlin) converter, which is part of the Kotlin plugin of the IntelliJ platform. Furthermore, Kotlin offers many convenient features for plugin development. In addition to null safety and type-safe builders, it also features lambdas which are extensively used by the IntelliJ Platform, like in Kotlin for Android. However, there are some aspects that the developer must pay attention to: the use of objects instead of classes to implement some *plugin.xml* declarations or the use of companion objects in certain scenarios. Another drawback is that the *GUI Designer*, provided by IntelliJ IDEA to easily develop interfaces, does not work with Kotlin because the binding between the XML file and the class properties does not work. Hence, all the interfaces present in this plugin have been made programmatically through the use of *Swing*[15].

General Plugin Structure

This section will highlight some components and features that are part of the plugins and the IntelliJ Platform. To have a clearer depiction of other elements it is recommended to refer to the official documentation.

The crucial components to understanding how a plugin works and how it interacts with the IDE are *actions*, *extensions*, *services*, *listeners*, and *extension points*.

[15] `https://docs.oracle.com/javase/tutorial/uiswing/`

[16] `https://www.jetbrains.com/help/idea/tool-windows.html`

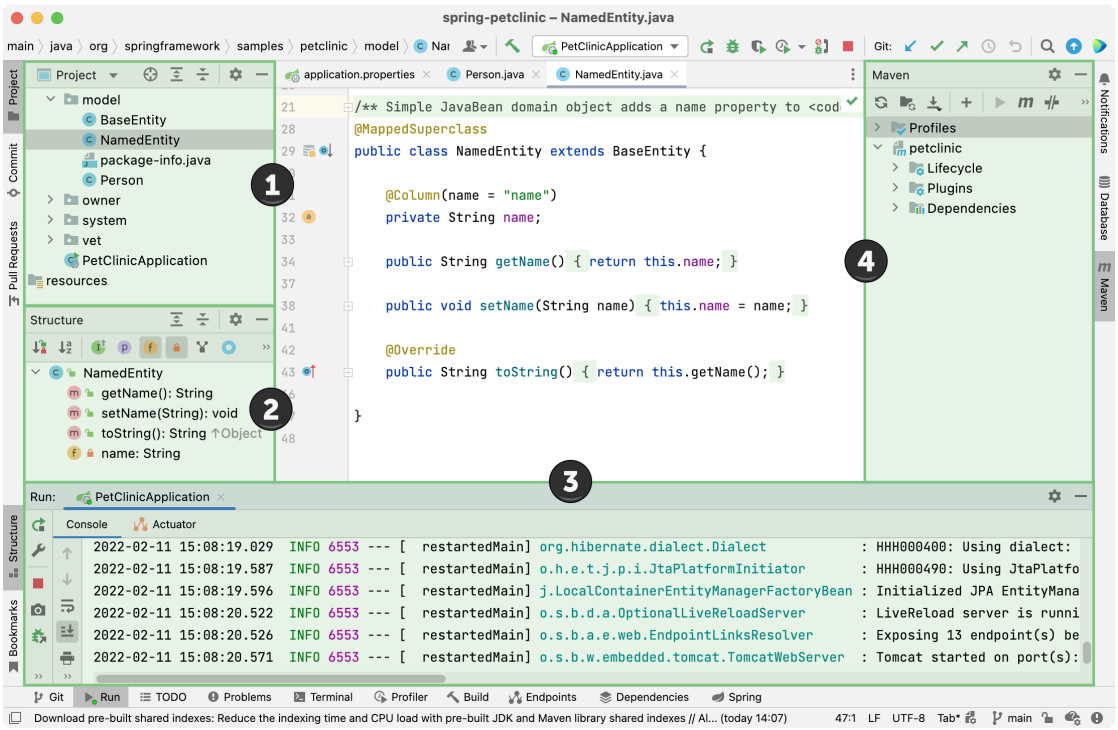

Figure 3.4: IntelliJ IDEA Tool Windows[16]

- Actions are the most common way a user can interact with the plugin. The user can invoke these functionalities from a menu, a toolbar, or through a keyboard shortcut. The actions can be organized into groups that can form a toolbar or a menu. An example of an action could be a search bar that appears when using the keyboard shortcut *CTRL + F*.

- Extensions are used to extend IntelliJ Platform's functionalities in the plugin in more complicated ways that cannot be solved through the use of actions with a menu or a toolbar. One of the most common extensions used is "com.intellij.toolWindow" which allows also plugins to have panels displayed at the sides of the IDE user interface as seen in Figure 3.4. Other extensions frequently used are the ones to add pages with options in the settings dialog of the IDE or the ones to extend language supports in custom language plugins. There are more than 1,500 extensions available both from the IntelliJ Platform and other plugins, allowing the developers to customize different parts of the IDE. The documentation offers, as mentioned before, a very convenient list where all available extension points and plugins that implement them[17] are

[17] `https://plugins.jetbrains.com/docs/intellij/extension-point-list.html`

listed. Alternatively, all the extensions are shown when developing the plugin through auto-completion in the IDE.

- Services are plugin components loaded when the plugin calls the *getService()* method. They encapsulate some reusable logic of the plugin because the IntelliJ Platform assures that only one instance of a service is loaded even if called several times. There must be an implementation class used for the service instantiation and if the service needs to do some cleanup when finished, the developer can implement *Disposable*. There are three types of services: application-level services, project-level services, and module-level services. Application-level services are a global singleton shared between the application, while the other two are instances shared between their scope (project or module). Additionally, some services can be *Light Services* if they meet some criteria. These services can avoid being registered in *plugin.xml*.

- Listeners allow the plugin to subscribe to events that are delivered through the message bus. They can be application-level listeners or project-level listeners. When declaring them in *plugin.xml* it must be specified the class that implements the service and the topic the listener wants to receive events from. Additional restrictions are possible, for instance allowing the listeners to operate only under certain operative systems or disabling them if the application is in testing mode. Listeners, like extension points, are all listed in the documentation.

- Extension Points refer conceptually to the extensions cited earlier. They are divided into two categories because, while extensions refer to functionalities extended by the plugin, extension points are defined in the plugin so that other plugins can extend the defined functionalities. Two types of extension points can be defined: interface extension points that allow other plugin developers to extend it with code, and bean extension points that allow other plugin developers to extend it with data.

The plugin's core, which defines the plugin structure, is the *Plugin Configuration File* (plugin.xml) available in the plugin resources folder. As was briefly already mentioned before, in this file all the previously listed components must be declared for them to work. Additionally, various information about the plugin and its components are listed in this file. To make some examples, there are defined the name of the plugin, the name of the vendor, the id and the version of the plugin, the id and the version of the targeted platform, the plugin dependencies, the actions, the extension points, the extensions, the project and application listeners, and, finally, the services.

Another concept that deserves to be mentioned is the PSI (Program Structure Interface). The PSI is the layer in the IntelliJ Platform that is responsible for

representing files in their syntactic and semantic code models. Through the PSI, a plugin can retrieve files and create them. Additionally, a PSI represents a tree of PSI elements and each can have as a child other PSI elements. The plugin can navigate this tree of PSI elements to perform various operations like renaming them, finding classes, finding all inheritors of a class, finding a superclass of a class, and more.

Finally, the plugin is built by using *Gradle*[18] or Plugin DevKit. The output will be a JAR file that contains the plugin configuration files, the classes that are needed to implement the plugin functionalities, and the plugin logo, if present. If the plugin has some dependencies, they will be as well bundled in the JAR file together with the plugin classes. To create a Gradle plugin project there are two options available. The first one is by using the "New Project Wizard" and selecting "IDE plugin" among the available generators. The second one, which is the one used for this project, is by using the available IntelliJ Platform Plugin Template on GitHub[19].

Plugin Implementation

The plugin was developed with the Kotlin programming language, as mentioned before, and the project was created using the GitHub template, the one specified in a couple of lines above. In general, the plugin project has been divided into seven folders to better separate different concepts: "actions", "communication", "dataClasses", "dataProviders", "enums", "listeners", "services", and "userInterface" as seen in Figure 3.5. Other than these, there is the resources folder which contains the plugin configuration file and all the icons used in the plugin interface.

A general idea of how classes interact with each other can be seen in Figure 3.6. A couple of classes have been omitted from this diagram since they offer only utility functions, one example being loading icons from the resources folder. Additionally, some minor associations between classes have also been omitted to avoid making the diagram too chaotic and complex to understand. A more in-depth analysis of what each folder contains and how the various classes shown in the UML class diagram interact between them will follow.

The content of each folder will now be explained in detail, along with details of each part that composes the plugin.

- The actions folder contains all the actions used in the plugin project. The definition and purpose of an action have already been discussed in the general overview of an IntelliJ IDEA plugin.

[18] `https://gradle.org/`

[19] `https://github.com/JetBrains/intellij-platform-plugin-template`

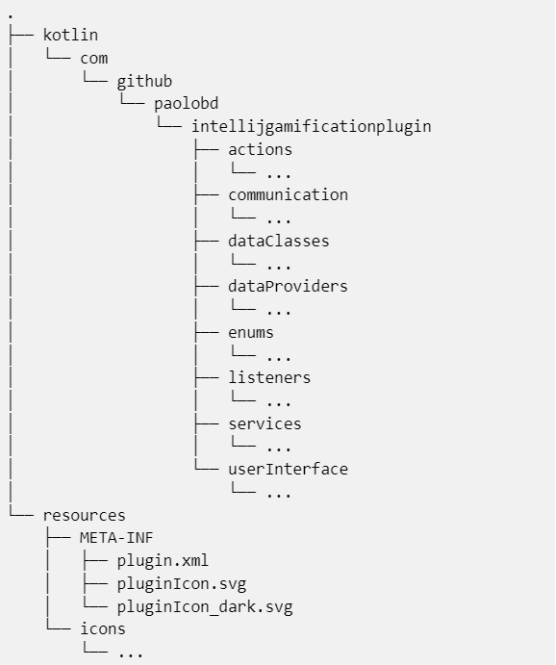

Figure 3.5: Gamification Plugin Project Organization

  - The "ResetAction" is the only action used in this project. Its goal is to reset the plugin's data, both the user profile and the achievements, if the user wants to. It is located in the settings menu of the plugin's window and it opens a dialog to confirm the user's decision.

- The data classes folder contains, as the name suggests, all the Kotlin data classes that are useful to the project's goal. Data classes are classes whose main purpose is to hold data. Additionally, the compiler automatically derives some members from their primary constructor like the equals or copy methods. All the «data class» that can be seen in the UML class diagram are in this folder.

  - The "Achievement" data class is used to represent the instance of an achievement with its characteristics such as name, description, and icon. Additionally, it contains info such as milestones, which is a list of numbers where each represents a step of the achievement. For instance, an achievement is unlocked when the action is first done five times, then ten times, and, finally fifty times. Along with this information, there is also a list of experience points that the user should receive for each step completed. This data class is primarily used by the enumeration classes to easily catalog the various types of achievements: global achievements,

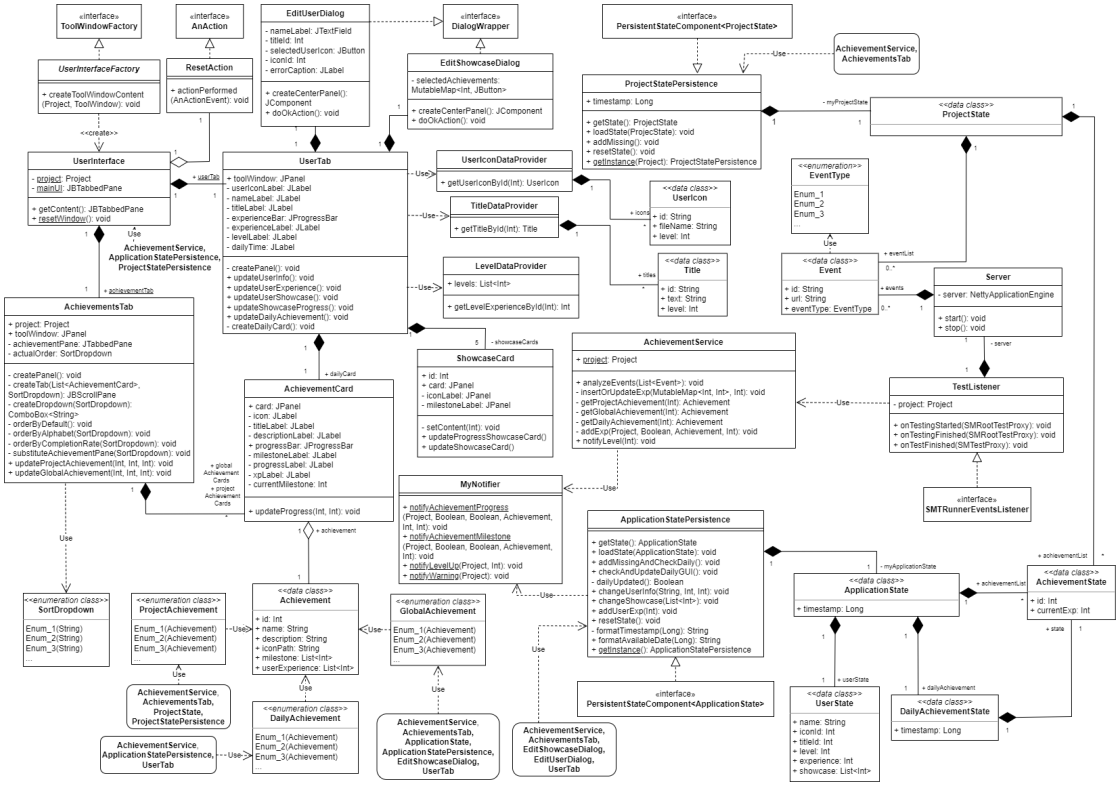

Figure 3.6: Gamification Plugin UML Class Diagram. See Figure A.1 for a bigger image

project achievements, and daily achievements. The decision to distinguish them in these three categories will be later explained in the gamification section. Other than for the enumeration classes, this data class is also used by the user interface to show the achievement information to the user.

– The "AchievementState" data class is used to keep track of the progress the user has made in a particular achievement. For this reason, the properties contained are the achievement id and the current user progress. This class is mainly used by the "ProjectState" and "ApplicationState" data classes. While the former contains a list of achievement states that refers to the project achievements, the latter contains a list that refers to the plugin's global achievements.

– The "Title" data class holds data of subtitles that the user can equip on their user profile. The class comprises two properties: the actual title and the level at which it unlocks for the user. More on this aspect will be analyzed in the gamification section. It is primarily used by the user

interface classes, which must show the available list of titles and the selected title to the user.

- The "UserIcon" data class is similar to the previous title data class. However, this class represents the icons that the user can equip on their user profile. It is formed by a string that represents the file name of the icon in the resources folder and the level at which it unlocks. As before, it is mainly used by classes of the user interface to show all the available icons the user can choose from and to show on their profile the one they have selected.
- The "UserState" data class is used to keep track of all the user information and its changes. It contains the name of the user, the id of the icon, and the id of the title they equipped on their profile. Additionally, it contains information on the level the user reached and their current value of experience points. The last thing present in this data class is a list of global achievement ids that the user decided to show off on their profile. This data class is mainly exploited by the "ApplicationState" class which holds the persistent data across the IntelliJ IDEA application. User interface classes also use this class to show correctly all the user information: icon, title, level, experience, and achievement showcase.
- The "ProjectState" data class stores the data the plugin needs that are relative to the current project. It contains a list of achievement states that represent the progress of the user in each project and an event list. The event list is formed by another data class, Event, which represents an action that the user has performed through Selenium WebDriver calls. The event data class is shared with the library that sends the events to the plugin, hence will be further analyzed and clarified later. The ProjectState data class is crucial for the plugin to work correctly and to accurately process the achievements' progress. Some achievements may require the user to execute different actions, hence it is fundamental to keep track of the past events. To further clarify this aspect, take as an example the achievement "Click five different objects". To work correctly, past clicks must be registered to check if a new click event was already executed or not. Overall, the ProjectState class is one of the cores of the plugin and it is mainly used by user interface classes to show the project achievements progress and by the "AchievementService" to perform the correct calculations to increase the achievements' statistics.
- The "DailyAchievementState" data class stores the information about the daily achievement. It contains the AchievementState data class and, additionally, a timestamp in milliseconds which represents the time the achievement has been created. This timestamp checks if the achievement

is still available or if a new one must be generated. It is part of the "ApplicationState" data class.

  - The "ApplicationState" data class is the other plugin's core. It stores the plugin information that is shared across the IntelliJ IDEA application. It contains the UserState class, which was previously illustrated, the achievement list that holds the user progress of the global achievements, and the DailyAchievementState class which contains information on the daily achievement. As the ProjectState class, it is used by user interface classes to show the global achievements progress and by the "AchievementService" that performs the operations to increase the achievements' statistics.

- The data providers folder contains three objects that hold static data. These objects are used by other parts of the plugin to obtain the data and utilize them in their implementation. A database to store this information could have been used, but since the achievements' details are stored in Kotlin enumeration classes and the project and application state are stored using the "PersistentStateComponent" interface, it seemed unnecessary to have an additional database only for this information. The database would have only added another layer to the project and unneeded complexity to store a very small set of information. Although, it is true that if in the future the size of the data will grow exponentially, it would be a wise decision to store them in a separate external database.

  - The "LevelDataProvider" object is used to store static information about the quantity of experience needed by the user to increase its level. For instance, to increase from level one to level two it is required to gain fifty experience points. Given that in this demo project, the maximum level is ten, it is clear that the set of static data stored here is very small. This object is used when calculations to add experience points to the user profile are made and to show the user the amount of experience they need to pass to the next level.
  - The "TitleDataProvider" object stores, instead, information about all the available titles to the user. At the moment, the user unlocks two titles per level. This object is used to show the user the list of available titles they can choose from and to return the correct title given its id.
  - The "UserIconDataProvider" object stores information about all the icons the user can choose its profile icon from. In total, there are thirty icons available to the user. This object is used to retrieve all the icons' source paths to show the icons and let the user select the one they prefer.

- The enumeration folder contains all the Kotlin enumeration classes. Enumeration classes can have a constructor, hence specific values can be passed to

it. The reason why it was decided to use enumeration classes to store the achievements was because they can be easily referred to in code. In this way, it is not necessary to use an id which could confuse the developers when they want to reference a particular achievement. By using enumeration classes, the IDE offers auto-completion of the available enumeration constants without risking referencing the wrong achievement or misspelling errors. Additionally, significant names can be given to the constants to recognize immediately which achievement they are referring to. For instance, "NUM_CLICKS" immediately conveys the idea the achievement counts the number of clicks performed.

  - The "ProjectAchievement" enumeration class contains constants and in each constructor is passed an Achievement class. It represents all the possible project achievements with their characteristics. It is used by the ProjectState data class to find the corresponding achievement given its id or to find the current progress of that achievement. Additionally, it is useful for the user interface classes to list all the possible project achievements and show their information.
  - The "GlobalAchievement" enumeration class is similar to the previous one but contains information about the achievements shared between the IntelliJ IDEA application. It is used by the ApplicationState data class and the user interface to list all the global achievements.
  - The "DailyAchievement" enumeration class, instead, contains only the daily achievements. It is used by the ApplicationState data class and the user interface to show information about the chosen daily achievement.
  - The "SortDropdown" enumeration class is used only to differentiate between the various means a user can order the global and project achievements in the achievements tab.

- The communication folder, as the name suggests, contains all the classes needed to enable the exchange of information between the gamification library and the developed plugin. Some of these classes are shared with the library to ensure the correctness of exchanged data.

  - "EventType" is an enumeration class that provides all the possible types of events that can be sent from the library. Examples of possible event types are "ELEMENT_CLICK" or "NAVIGATION". It is used in the Event data class and it is fundamental to discern which achievement progress should this event account for.
  - "Event" is a data class that holds different properties to ensure that it is possible to state if two events refer to the same Selenium WebDriver action or two different ones. The properties of the event data class are an

id, which represents the DOM object clicked on the web page, the URL, which represents the URL in which the action has been executed, and the EventType. These three elements together allow us to discern between different events and, consequentially, to correctly increase the progress of the various achievements. It is mainly used in the ProjectState data class to keep track of all the Selenium WebDriver events that have occurred in a project and in the achievement service, which makes all the calculations to increase the achievements progress.

  - "Server" is the class that is responsible for starting and stopping the server where the information is sent from the library. *Ktor*[20] is the framework used to build framework applications. It has been selected because it is highly scalable due to the use of coroutines, it is developed by JetBrains, and it offers easy-to-use APIs. Additionally, its setup is straightforward. The server is run and closed at specific moments when it is known that the plugin is going to receive events from the library. This decision has been made to prevent the server from wasting resources: it would have been useless to keep the server running from the start of the plugin until the closure of the IDE. Additionally, it would have been more prone to errors and disconnections. When the library sends a list of events through the REST POST API, the server saves it, and then will later be analyzed by the achievement service.

- The listeners folder contains all the listeners used by the plugin project. In this project, there is only one listener: the "TestListener".

  - The "TestListener" is a listener that receives messages from the "SMTRunnerEventsListener" topic. To take advantage of this topic, it is mandatory to implement the "SMTRunnerEventsListener" and override the needed methods to achieve the project's goal. Briefly, this interface allows the plugin to be notified when certain actions are executed from the user's IDE. In particular, it monitors all the events regarding testing and unit testing: when the user executes some tests on their project and the plugin is installed, the plugin's listener will be notified and some actions can be performed by overriding the interface methods. For instance, the method "onTestingStarted" is called just before the testing starts, before tests and suit launching. The TestListener component is crucial because it allows the plugin to know when it is time to check for the events sent by the library and to increase the achievements' progress accordingly.

[20] `https://ktor.io/`

To go into more details of the project implementation, in the "onTestingStarted" method the server is started. It is started at this moment because, if the user is using the library in their application, soon the plugin will receive some events. The server is only stopped when the testing finishes, which is in the "onTestingFinished" method. In this way, the plugin avoids wasting resources by starting the server only when it is supposed to receive information from the library. The captured events from the server are analyzed in the "onTestFinished" method through the achievement service. It was decided to analyze events at the end of each test and not at the end of the whole testing suite to have a more responsive plugin. If the plugin does not show progress for a long time, the user could think that it is stuck and their events are not being registered. In this way, they have more immediate feedback on what is happening. Finally, some other methods available in the listener class can be used to monitor other activities. For instance, the method "onTestFailed" to keep track of the number of tests failed and their names, or the methods "onSuiteStarted" and "onSuiteFinished" to know how many tests are in the suite executed by the user.

- The services folder contains all the application and project services that have been used in the plugin. The plugin uses, in total, three services: the "AchievementService", the "ApplicationStatePersistence", and the "ProjectStatePersistence" service, as also seen in the UML class diagram.
  - The "ProjectStatePersistence" service implements the "PersistentStateComponent" interface. It allows to persist state classes, giving flexibility for the values to be persisted, their format, and their storage location. It is the API provided by the IntelliJ Platform to store simple key-value entries or more complex state classes between restarts of the IDE. When implementing the interface, the developer can specify the state class to persist and specify the storage locations. Additionally, they can add a flag to indicate that the plugin needs to be reloaded when the state changes. The state classes are typically saved in an XML file, and the location of that file, for the project state persistence, is by default in the project's *.idea* folder. The project state persistence service has been implemented at a project level and it stores the ProjectState data class.

    At that moment, during the development process, some doubts arose. If the value is stored in an XML file inside the *.idea* folder, the user can freely modify its content or, by accident, delete the file. This downside was taken into account while developing the plugin. While it is true that the state class stored in that way is not entirely reliable, it is also true that this way of implementing persistence is very straightforward and it

does not add much complexity to the whole project. Furthermore, this plugin has been made only for study purposes and its final goal is not to be publicly available in the IntelliJ IDEA Marketplace. The users that will try this plugin will be made aware of the consequences that will show up if they delete the file, and they will be incentivized to back up the file if they are afraid of losing their progress. However, it is important to keep in mind that plugin testing sessions with the users will not be very long, hence the risk of problems is very minimal.

Regarding the possibility of the user modifying the file and compromising the results, it has also been kept in mind. It would not make sense for the user to modify the value of their achievement progress. There is nothing at stake in this context, the plugin is only made to incentivize themselves to do a task that, otherwise, would be tedious and monotonous. Furthermore, the absence of a global leaderboard or compensation based on performance reinforces the idea that a user would not modify their progress file. If, in the future, the choice is to publish the plugin it would make more sense to use an external database. An external database would also allow for additional features such as a global leaderboard. Collecting all the users' information in a single database would permit the plugin to show the progress and the profile of other users. An internal database could not be used to store the user's information because the plugin will be bundled in a jar file, hence it does not permit internal modifications. A read-only database would have been possible, but also not effective in storing dynamic user and achievements information.

Returning to the implementation discussion, this service is used by other classes when it is needed to retrieve its instance to check the saved data of the project state class. For instance, when showing the current achievements' progress in the project achievements tab, or when checking the WebDriver events to decide which achievements have advanced.

– As before, the "ApplicationStatePersistence" implements again the "PersistentStateComponent" interface. Hence, the interface will not be further explained in this section, but this time it is an application service. Being an application service means that the state is shared between the IntelliJ IDEA whole application, thus between every project opened. That is why this service stores the ApplicationState data class: it stores information about the user such as name, title, icon, and information about the progress of the global achievements which are shared between all projects. This time, the file is saved in a folder less visible to the user so it is less prone to modifications or accidental deletion.

  This service is used by other classes when it is needed to retrieve its

instance to check the user's information to show them on the interface, to show the correct global achievements' progress, or to know which is the chosen daily achievement. Moreover, this service calls the methods to update the interface when new information needs to be reflected.

- The "AchievementService" is the service that is in charge of analyzing the received events from the server and performing the computation to increase correctly the progress of the achievements. The events are passed to the achievement service through the test listener that manages the server. The "analyzeEvents" method receives the list of events as a parameter and, after checking if the event is already in the project list of registered events and checking its event type, it determines how many progress points each achievement should receive. Finally, after analyzing all the received events, the service updates the progress value in the persisted states and then calls the functions to update the user interface, to reflect the new progress.

- The user interface folder contains all the classes that are used to create and update the tool window of the plugin. As briefly already mentioned during the choice of the programming language, Kotlin, unfortunately, does not support the GUI Designer of IntelliJ. Thus, every part of the user interface has been implemented programmatically by using Swing. Through the use of the different layouts available and the various GUI elements present, it has been possible to create a pleasurable interface for the user that follows the canon and the styles of the IntelliJ IDEA platform.

  - The "UserInterfaceFactory" implements the interface "ToolWindowFactory" which is one of the extension points already illustrated in the general plugin interface section. This interface allows the plugin to have its window displayed to the side of the IDE so that the user can have visual and more responsive feedback on its progress. Its purpose is to initialize the "UserInterface" class and put its content in the tool window.
  - The "UserInterface" class initializes a tabbed pane and adds as tabs the "UserTab" class and the "AchievementsTab". This class is also the way through which other plugin components access the two tabs and other user interface elements to update them.
  - The "UserTab" class is used to visualize the user's information and the daily task. This is all data that can be acquired through the Application-State class. In particular, the user information includes their icon, their name, their title, their level, and the experience they have accumulated. Additionally, there is a showcase where user can select up to five global achievements to show off on their profile. Finally, there is information

regarding the daily mission the user can accomplish. Through two edit buttons, the user can edit their user information and edit the achievements they want to show off.

- The "ShowcaseCard" class represents the single global achievement that the user can put on their profile. It has as properties the id, the icon, and the current progress of the chosen achievement. It is used by the UserTab to keep track of what achievements are shown on the shelf and to update the progress if needed.
- The "EditUserDialog" implements the "DialogWrapper" interface and it is used to let the user edit their name, title, and user icon. When the edit button in the user tab is clicked, this dialog appears and it displays to the user all the available choices they have. At the bottom of the dialog, there is a button to save the changes or to cancel them.
- The "EditShowcaseDialog" implements the "DialogWrapper" interface and it is used to let the user choose which global achievements they want to display on their profile. As the other dialog, at the bottom, a button is present to save the changes and another one to discard them.
- The "AchievementsTab" class is used to create the second tab where information about both global and project achievements is shown. The tab itself is divided into two tabs, one for the global achievements and the other for the global achievements. Additionally, a dropdown is available to let the user sort the achievements in the order they prefer. To represent the information about the singular achievement another class is used called "AchievementCard". The achievements tab has, among its properties, two lists of achievement cards. One represents the global achievements cards and the other represents the project ones.
- The "AchievementCard" class represents the single achievement interface instance seen in the two achievements tab. It receives the achievement's data and displays its icon, name, description, the user experience gained by completing the achievement, the milestone at which the achievement is, the current progress, and the total progress needed. The current progress is retrieved from the project and application state classes which hold the current achievement value. This class contains also a method that, given the amount of experience, updates the progress of the achievement in the visual interface.
- The "MyNotifier" class is not related to the tool window but it still is an important part of the user interface. It exploits the notification that the IntelliJ IDEA IDE can create to notify the user about some plugin events. When an achievement is completed a notification pops up to let

the user know that some experience has been added to their profile. A notification also appears when a certain percentage of the achievement has been completed. Furthermore, the user is also notified when a new milestone has been reached or when they have leveled up.

This concludes the in-depth description of the plugin's components. To briefly summarize, the core of the plugin is the TestListener which creates a server before the testing begins in the user application. When a test finishes its execution, the events sent to the server by the gamification library are analyzed by the AchievementService. The AchievementService calculates the correct amount of progress each achievement has reached. The user interface, which is displayed in a window at the side of the IDE, is then updated accordingly with the new achievements' progress. The user can also modify the look of their user information through the interface. All the data, which are the achievements' progress, the events, and the user information, are stored in a project and application state data class implemented through the persistent state component interface.

Overall, the development was not an easy task. Developing a plugin requires the developer to know some of the mechanisms inside the IDE. It is crucial to understand what is possible and what is not possible to do, and how much effort it requires. The IntelliJ Platform SDK documentation is available but the examples it provides are straightforward and do not go into much detail. The developer is required to examine how other plugin developers implemented their plugins and also to examine IntelliJ IDEA's internal code on GitHub. The forum on the JetBrains website and the Slack channel are also optimal tools for exchanging ideas with other developers and asking for help. Finally, the creation of a compelling user interface for the user that maintained the general theme of the IDE also required some planning. The impossibility of using the GUI Designer with Kotlin slowed the development of the user interface because it made necessary the use of Swing, thus developing the user interfaces only programmatically without the aid of an external designer tool.

Plugin User Interface

User Interface plays a crucial role in the development of software, influencing how users react and perceive the applications. In the context of a plugin for IntelliJ IDEA, a well-crafted user interface is essential for boosting productivity and enhancing the user experience. Additionally, since the plugin is about gamification, a good interface encourages the user to utilize it and helps in highlighting their accomplishments.

The user interacts with the plugin through a tool window, which is a separate window that appears on one of the sides of the IDE. Since the plugin is built for

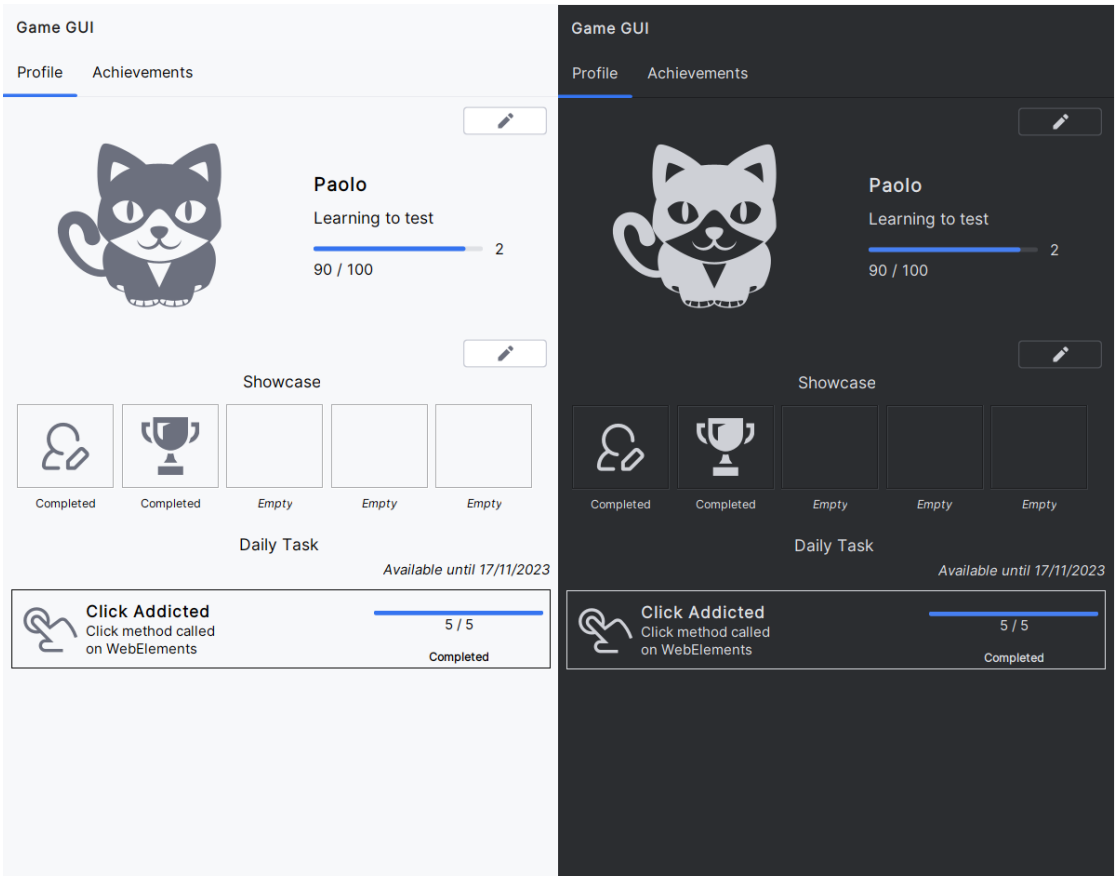

Figure 3.7: Profile Tab

IntelliJ IDEA, some conventions[21] had to be respected and followed to make the plugin feel like part of the IDE and not something too distant from its design. The user interface has been developed by following the style of the new User Interface (UI) recently released for IntelliJ IDEA, which changes its themes, icons, and fonts. In particular, to please the largest possible audience, the plugin has been designed in two themes: a light one and a dark one. The two main aspects that need to be visible in the plugin by the user are the information regarding the user and the progress of the achievements. To achieve that, two main tabs are positioned in the plugin's window: the "Profile" tab and the "Achievements" tab.

The profile tab, as can be seen in Figure 3.7, displays the user page with the information associated with it. On the top part, there is the icon that the user chose for their profile, the name, the selected title, and the user level together with the

[21] `https://jetbrains.design/intellij/`

progress bar that represents the user's advance in the current level. Immediately below, there are five squares where the user can put five achievements to display on the profile. Under their icons, a label that describes the state of the achievement is shown: if it is not completed, the current progress is displayed. Finally, a daily task is on the bottom that shows the daily assignment that the user has. The needed progress to complete it, the user experience it gives to the user profile, and its availability time are shown together with the achievement's information. As can be seen if Figure 3.8 and Figure 3.9 both the user information and the showcase are freely customizable by the user in different dialogs that open when clicking the respective edit button.

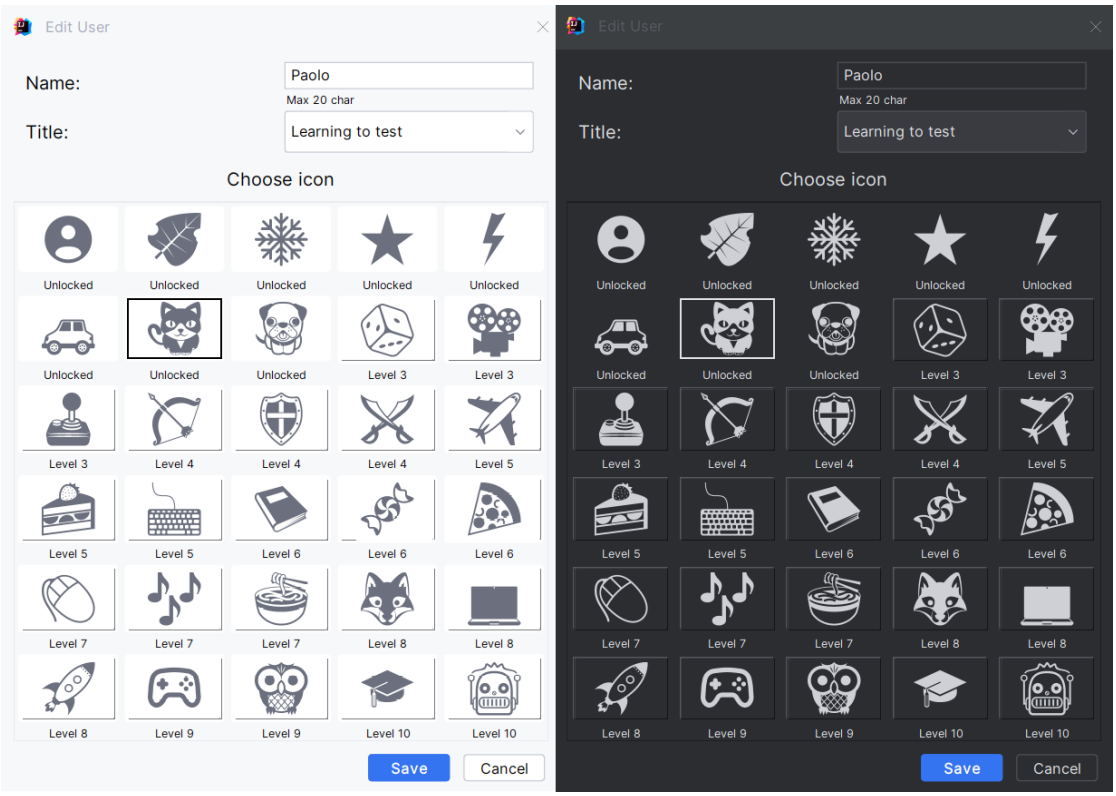

Figure 3.8: Edit User Dialog

The achievements tab, as shown in Figure 3.10, exhibits all the available achievements and the progress the user has reached for each of them. This tab is further divided into two tabs to differentiate between the achievements that are shared between all the user's projects and the achievements that are project-specific. The visuals between those two tabs do not change, only their content. On the top right, there is a dropdown menu that the user can use to sort the achievements to their liking. The available sort options are default order, alphabetical order, and by

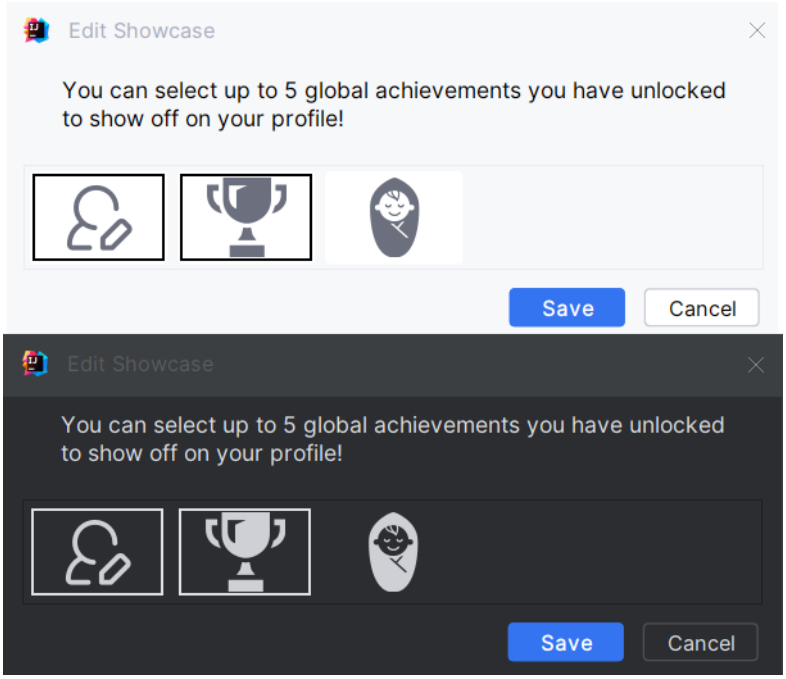

Figure 3.9: Edit Showcase Dialog

percentage of completion as seen in the figure. Each achievement card contains the achievement's information such as the icon, the name, and the description. Additionally, it displays the milestone the user has reached on that particular achievement and its progress. If the achievement has been completed, it gets highlighted with a special border to make it stand out to the user. With this layout, all the information the user needs to know about is shown in a compact way that is immediately recognizable and displays only the details the user is interested in.

IntelliJ IDEA's notifications are also used in the plugin to provide the user with useful information. Every time the user levels up, an achievement is completed, a significant milestone is reached or a problem arises, the user is notified by exploiting the notifications offered by the IntelliJ IDEA's environment.

Java Swing has been the primary source to build the plugin's user interface. Since the GUI Designer was not available to use with Kotlin, all the interfaces have been built programmatically. This involved the understanding of the various layouts and components that populate the Swing library and how they interact with each other. Hence, building a well-crafted interface that satisfies the plugin's and the user's needs took a good amount of time and effort. But this aspect was considered essential to bring to the user an enjoyable experience that also enhances gamification features. Having a good interface further highlights the accomplishment the user reaches and encourages them to keep using the plugin.

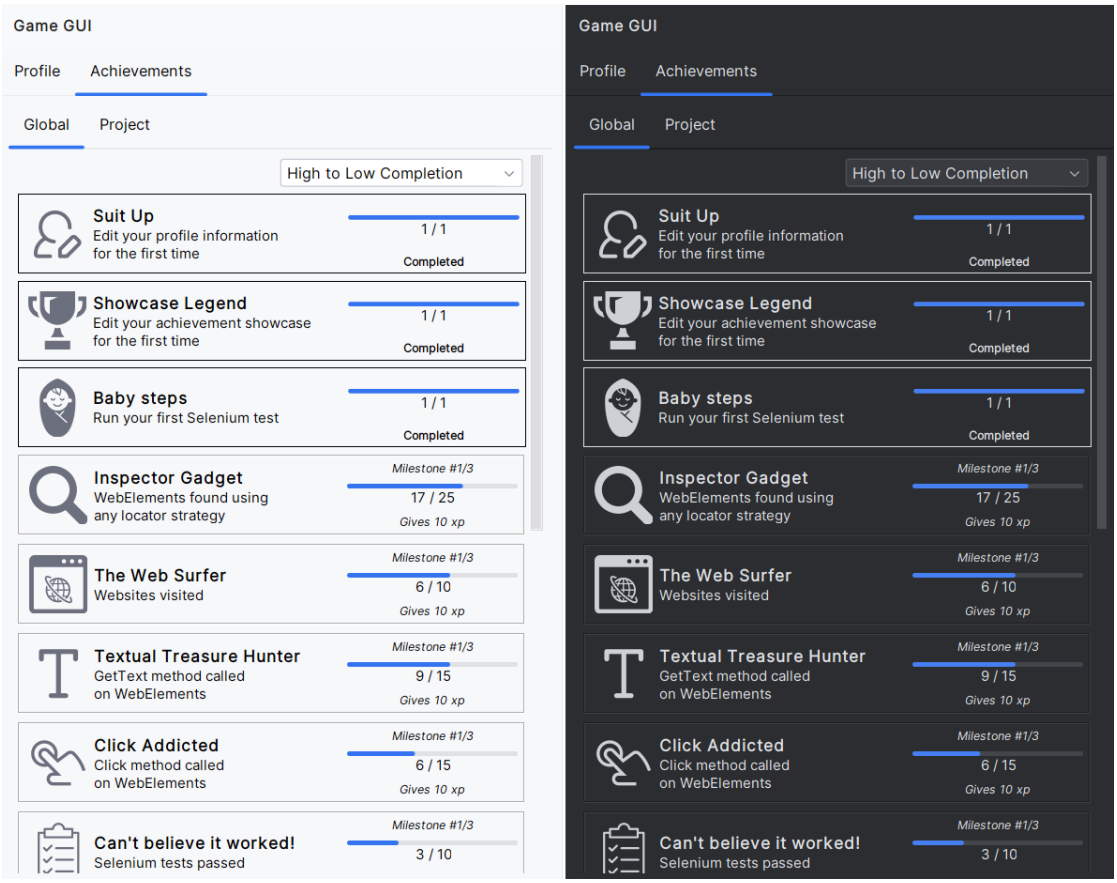

Figure 3.10: Achievements Tab

3.2.3 Intercepting Selenium WebDriver Calls

The other crucial topic to discuss was letting the plugin know which actions the user executed through Selenium WebDriver during the web application testing. After studying Selenium's documentation, an eligible candidate was found. Using the WebDriverListener[22] seemed like an optimal approach to this goal. The WebDriverListener is a public interface available through Selenium that has methods that allow the developer to execute some code before and after each Selenium WebDriver call. It needs to be attached to the WebDriver instance by using the "EventFiringDecorator" class. For instance, the "beforeClick" and "afterClick" are some of the accessible methods, and in this particular case, they allow the developer to perform operations before the WebDriver clicks an element and after

[22] `https://www.selenium.dev/selenium/docs/api/java/org/openqa/selenium/support/events/WebDriverListener.html`

the WebDriver clicks it. Many more functions are available and they permit the developer to react to almost every Selenium WebDriver call. A list of these can be found in the Selenium's documentation cited before.

At first glance, it seemed like the issue was already solved, but it soon became clear that another important question needed to be discussed: how to decorate the user's WebDriver instance with the developed implementation of the WebDriverListener. Several options were discussed:

- Using aspect-oriented programming via AspectJ[23]. With AspectJ, it is possible to define aspects to intercept WebDriver calls by adding custom logic. Hence, just after the creation of the WebDriver instance, the WebDriverListener could decorate it. It works with existing code, reducing the need to modify existing source code. The disadvantage is that is hard to set up the environment. The user could be asked to do it by following a provided guide, but it would require non-trivial changes to their project and build environment, defying the purpose of creating a gamified environment as straightforward as possible. If the setup is perceived as difficult, the user is less incentivized to do it with the sole purpose of adding gamification elements. Another solution would be letting the plugin automatically set up the user environment. This possibility is still not optimal, because the risk of messing up with the user's environment is very high given that the details of the user's project are not known.

- Exploiting bytecode manipulation. This option gives the developer full control of the implementation and it can be extensively customized. It allows direct integration with the internals of the WebDriver, enabling access to all its methods. The disadvantage is that it is complex and requires knowledge of Java class loading and bytecode manipulation. It can also easily introduce instability if not done carefully. Finally, the setup problem introduced in the previous solution still applies in this context.

- Defining a proxy server. It was considered because it separated distinctly the user's application and its WebDriver calls from the plugin code. The disadvantage is that it adds complexity and adds a layer to the testing setup. This can cause latency issues, affecting the speed of test execution. Additionally, it gives less control to the WebDriver calls than bytecode manipulation and aspect-oriented programming.

- Taking advantage of IntelliJ Platform capabilities. The idea was to monitor, in some way, the WebDriver instance and decorate it with the WebDriverListener during runtime. It became clear that this is not possible: IntelliJ Platform's

[23] `https://eclipse.dev/aspectj/index.php`

plugins support mostly static analysis and not program execution. Monitoring the program execution would be possible only if the user was using the debugger.

Another idea taken into consideration was to exploit the PSI to find, in the file, where the WebDriver instance is created and, below it, add the code to decorate it with the WebDriverListener. This would have been feasible, but adding some lines of code without the user realizing it and adding one file in the project where the WebDriverListener was defined seemed not a good solution.

- Developing an external library. Together with the plugin, an external library would have been given to the user that contained the definition of the WebDriverListener and the setup of the WebDriver. A small guide would accompany the library to explain how to import it and how to take full advantage of it.

The best compromise seemed to develop the additional library. The user would still have full control of their project deciding when to use it and when not. Furthermore, adding a library as a dependency is not a hard task, and, giving a guide, there is a small risk of possible issues. Indeed, this process is not completely transparent to the user as one would expect from a gamified environment. But still, it seemed the best trade-off given its ease of being incorporated into the user's application and the very few lines of code that need to be added by the user: just one after the WebDriver creation. The other solutions were arduous to implement and they messed too much with the user application settings.

3.2.4 Gamification Library

Once figured out that creating an external library that communicates with the plugin was the best choice, the development began and it did not have any particular issues. The main classes of the gamification plugin are "GameGui" and "GamifiedListener". Additionally, there are the Event class and the EventType enumeration that are shared with the plugin to exchange information as seen in Figure 3.11.

The GameGui class has a static method that takes as a parameter the WebDriver instance used by the user. Here, the WebDriver gets decorated with the GamifiedListener and returns the decorated driver to the user. The user can then continue to use their driver as before as seen in Listing 3.1.

The GamifiedListener implements the WebDriverListener interface. When certain WebDriver methods are called, the GamifiedListener registers the events. For instance, in the "afterClick" method, the listener adds to the list a CLICK event, specifying the object that was clicked and the URL of that object. As a reminder, this is the Event class, which has as properties an id, a URL, and an

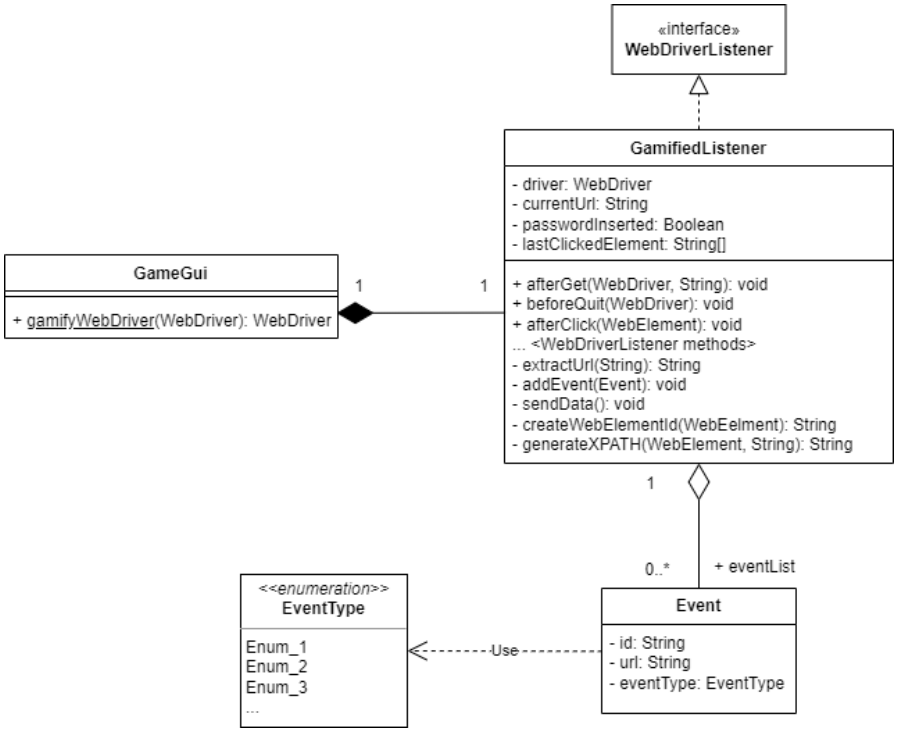

Figure 3.11: Gamification Library UML Class Diagram

EventType. At the end of the test, when the WebDriver calls the quit method, the events are sent to the server opened by the plugin at the start of the testing session.

Listing 3.1: Attaching the GamifiedListener to the WebDriver

```
1    ...
2    driver = new ChromeDriver();
3    driver = GameGui.gamifyWebDriver(driver);
4    driver.get("https://www.polito.it/didattica");
5    ...
```

Some issues that needed to be resolved appeared. The first one was what to use as an identifier for the object used by the WebDriver method. The final decision was to take, in order, the first of these properties that is not null or empty: id, name, and XPath. The ideal solution would be to take the id but since not all web elements have an id, there had to be alternatives. To create the XPath a defined method is used to generate it from the element's webpage.

The second issue is that not all sites are built the same, by the same programmers or using the same conventions. Hence, some compromises had to be made for the tracking to work. To make an example, the "afterClick" method gets called

when the WebDriver clicks an object, even if it is not a button. Since there is no reliable way to know if an object is a button because not all of them have their type attribute specified as a button, the decision was to assume the user would only click things that are buttons during testing. A similar logic applies to check if a login was successful. After entering the credentials and clicking the button to log in, there is no reliable way to know if the login was successful or not. Some websites redirect the user when they log in, others do not. Another possibility would be to check if there are some user-specific properties, such as their name, on the newly loaded page but, still, this is not a valid technique. Given all the variegated sites present on the internet, and the different programming styles, some compromises had to be made on what is possible to track with events and which achievements to develop.

3.2.5 Data exchange

The data exchange is essential for the gamification plugin to work as intended. The external library sends the data regarding the WebDriver calls to the plugin so that it can compute correctly the achievement progress of the user. It is accomplished, as explained in the previous sections, through REST APIs that exchange data using the JSON format. The server is created and managed by the plugin through the Ktor framework. The gamification library sends through a POST REST API the list of intercepted events in JSON format. Upon receiving the list, the plugin analyzes the list of events through its service.

The essential part of this process was understanding the best time to start and stop the server. Keeping running the server for the entire duration of the plugin did not seem like a good approach, especially because the user is not always executing tests. Before executing them, they have to write the code of the tests which could take some time. Additionally, sometimes the IDE is left open without directly working on it, and keeping a server running would be a waste of resources. Hence, the server is started through the plugin's test listener before the test suit starts. At the end of each test, the plugin takes all the events received by the server and starts to analyze them. Finally, when the test suit ends, the server is stopped. The initialization of a server at each test suite slows the time before the actual testing starts, but it was considered a better option than keeping a server constantly in execution. Additionally, Ktor is a lightweight web framework, hence the initialization should not take too much time.

The other important aspect is to synchronize the library to be sure that it sends the list of events when the server is running. Since the user is using a WebDriver to test the web application, a good moment to send the information seemed to be before the call of the quit method of the WebDriver. Hence, in the "beforeQuit" method of the WebDriverListener, the data gets sent by opening an

HTTP connection. At that moment, certainly, the server is still open, because it gets initialized before the start of the test suite. Also in the last test, it is secure that there will not be any issue: the server closes after the last test finishes but the event list gets sent before the quit method of the WebDriver is called. Since the quit method is done before the test ends, the information will be sent before the server in the plugin stops running.

The data that is exchanged, as briefly mentioned before, is a JSON that represents the list of events. As seen in the previous section, an Event is a class that has as properties an id, a URL, and an EventType to uniquely identify it. The EventType is the type of action that is performed by the WebDriver and is tracked by the plugin's achievements. For instance, the EventType could be "CLICK" or "LOGIN". The ID identifies the object in the web page that is the target of the WebDriver call and the URL is the site of the page in which it is found. An example of a JSON list of events can be seen in Listing A.1.

3.3 Gamification Elements

The other crucial part of the work of this thesis that was conducted along with the development of the plugin is the choice of gamification elements suitable for this particular context. The gamification elements should motivate the user to perform in-depth scripted GUI testing, mitigating the sensation of repetitiveness and boredom that often comes with it. The chosen game mechanics are profile customization, progression, unlockable content, achievement showcase, achievements, and daily tasks.

3.3.1 Profile Customization

The user can customize the appearance of their profile page (Figure 3.7 and Figure 3.8), which includes changing the user name, icon, title, and achievement showcase. Having the chance to customize the profile enhances the feeling of ownership and identity, stimulating the user to keep using and interacting with the plugin. Additionally, the presence of unlockables further reinforces a positive feedback loop. As will be further explained later, the unlockable contents of the plugin are icons and titles. The possibility of equipping rare content that also symbolizes user progression creates a stronger bond between the user and their profile page. It is a chance to show the world their accomplishment and how they want to represent them. The Octalysis core that represents avatar customization is **Ownership**.

3.3.2 Progression

On the user page (Figure 3.7, along with the user information, there is also a user level and a progress bar. The progress bar represents the current progress of the user at the current level. The presence of explicit progress bound to the profile is another way for the user to measure their accomplishments and how much they have interacted with the plugin. Having it displayed at all times is a positive reinforcement for the user to keep completing achievements and keep testing the web applications. Reaching higher levels and seeing the growing progress should also stimulate the users to perform more in-depth scripted GUI testing. The implemented levels and their experience in this first draft of the plugin can be seen in Table 3.1. The level ten has been chosen as the maximum level the user can reach. In the future, the needed experience and the number of levels can be easily modified to adjust the speed of progression, which can be affected by the addition or deletion of achievements. The user levels up by gaining experience points rewarded when the achievements are completed. The Octalysis core drive that represents progression is **Accomplishment**.

Level	Experience to progress
1	50
2	100
3	200
4	300
5	400
6	500
7	600
8	700
9	800
10	9999

Table 3.1: User Levels

3.3.3 Unlockable Content

Unlockables, as briefly cited before, are a core part of user customization. The unlockable contents in this plugin are user icons and user titles (Figure 3.7 and Figure 3.8). An icon is an image that the user can equip to represent themselves and a title is a short phrase that appears under the user name that spells out a funny or catchy phrase to describe the state of the user. The list of titles available in the gamification plugin can be seen in Table 3.2. These elements are unlocked when the

Title	Level unlocked
Novice	1
Always asleep	1
Learning to test	2
How does it work?	2
Now I get it!	3
GUI Apprentice	3
...I don't really get it	4
Gamification is fun	4
Bug bounty hunter	5
Never give up	5
Testing artisan	6
Master debugger	6
Web Wizard	7
Code Ninja	7
UI Virtuoso	8
Selenium Connoisseur	8
Gamification Enthusiast	9
Game Changer	9
GUI Testing Wizard	10
The cake is a lie	10

Table 3.2: User Titles

user reaches certain milestones represented by the user levels. For instance, when the user reaches level three they will unlock three new additional icons to equip and two new titles. In this way, rarer content that is unlocked at higher levels acquires greater value to the user, stimulating them to complete more achievements and perform more testing. The Octalysis core drive stimulated by unlockable content is **Ownership**.

3.3.4 Achievement Showcase

The achievement showcase is a way to allow the user to further highlight their accomplishments. The showcase consists of the possibility of the user to equip up to five global achievements they have completed on their profile (Figure 3.7 and Figure 3.9). In this way, they can always update the showcase to show off rarer achievements or the objectives they are most proud of. This adds a further layer to user customization and identity and it stimulates the user to unlock rarer

achievements and reach higher milestones to show off on their profile and share with the people around them. An achievement is showcased by its icon and under it there is a label that specifies the current progress of the achievement or if it is completed. The Octalysis core drive represented by the achievement showcase is **Social Influence**. Even if users cannot look at other players' profiles, there are still other ways to share the achievement showcase outside the system. For this reason, it still has been considered appropriate to consider the Social Influence drive for this gamified element.

3.3.5 Achievements

The achievements (Figure 3.10) are the foundation on which the other gamification elements of the plugin are built. An achievement is represented by an icon, a name, and a description. The icon and the name have been carefully selected to make them memorable and easily recognizable by the user. As titles, they often have pop culture references to create a deeper bond between the user and the plugin and make the experience more enjoyable. The description, instead, is necessary for the user so that they are aware of what they have to do to progress on specific achievements.

An achievement has also different milestones that when completed award experience points to the user. These points sum to the user's current progress and may trigger a level-up which, consequently, unlocks titles and icons. Hence, completing achievements is strictly bound to the user's progress and the possibility of unlocking content. Completing milestones also offers the user the possibility of equipping achievements on the showcase section, as also explained before. Milestones are how the achievements are divided into several steps. Each step is represented as a milestone and each milestone awards to the user a custom amount of user experience. For instance, the achievement that keeps track of the selenium tests passed has three milestones: one that unlocks at five, one at ten, and one at twenty passed tests. Typically, in the gamification plugin, the milestones are one or three but, in the future, this can be customized to rectify some inconsistencies in the progress. An achievement may take too much time to complete than needed, or it may be completed too fast by the users.

Finally, achievements in this plugin have been divided into two categories: global achievements and project achievements. The project achievements are related to the specific IntelliJ IDEA project while the global achievements are transversal along the whole IntelliJ IDEA application. Hence, global achievements have higher milestones than project achievements because they are related to all the projects the user uses the plugin on. Some global achievements are a repetition of project achievements but are more difficult to complete. Some requirements that are present among the project achievements do not appear in the global ones because it has been

considered they were too specific to put them in this category. This decision has been taken to stimulate the user to complete as many achievements as possible when interacting with a specific project. With project-specific achievements, they have a measure to understand if they still need to explore and test the web application or if they did enough. Additionally, having project-specific achievements gives constantly to the user a sense of progress. With only global ones, the milestones would have been many more with higher requirements and, reached a certain advanced point, the user may feel like they are not progressing because they unlock a new milestone very rarely. Adding project achievements gives an immediate sense of progress when testing new web applications because the requirements of the initial milestones are low and easily achievable.

The list of the project achievements can be seen in table Table 3.4. The majority of them are counters of specific actions that are executed with the Selenium WebDriver or actions that are executed on WebElements. For instance, there is the navigation achievement that counts all the websites the user has visited, or there is the achievement that counts the number of times the user has clicked on different WebElements. Apart from the achievements that are related to Selenium and scripted GUI testing, some more general ones keep track of the number of achievements passed using Selenium or the number of achievements that have been fixed after failing.

The global achievements are similar to the project ones. Some achievements keep track of the interaction with the Selenium WebDriver and WebElements actions like the click or the number of websites visited in total. Additionally, the achievements that count the number of Selenium test passing are still there but they are global and count all the projects. Some particular achievements have been added in this section to introduce the user to various features of the gamification plugin. For instance, there is an achievement that rewards the user the first time they modify their profile page or achievement showcase. These achievements stimulate the user to discover all the possible features of the plugin and become acquainted with them. There are also some rewards for the first Selenium test run or the first test not passed to give instant gratification to the user and instantly engage them in the use of the plugin. A list of the global achievements that are available in the plugin can be seen in Table 3.5.

The final element that has not been discussed yet is the daily task. A daily task is an achievement that must be completed in a limited amount of time, meaning before the end of the day. Each day a random task gets picked to stimulate the user to utilize the plugin daily and, also, if the user has already completed all achievements or their progress has become too slow, they still have some way to advance easily through the levels. The complete list of daily tasks defined in the plugin can be seen in Table 3.6. They are similar to the corresponding global achievement, the only differences are the threshold on which they reward the user

with experience points.

Both global and project achievements represent the **Accomplishment** core drive of Octalysis, meanwhile the daily task is most aligned to the **Unpredictability** core drive. The reason is that the daily task is random, hence the user is incentivized to discover what each day has to offer them.

3.3.6 Octalysis Framework

To summarize, the introduced gamification elements and their associated core drive are shown in Table 3.3.

Gamification Element	**Core Drive**
Achievements	Accomplishment
Daily Task	Unpredictability
Profile Customization	Ownership
Unlockable Content	Ownership
Progression	Accomplishment
Achievement Showcase	Social Influence

Table 3.3: Plugin Octalysis

Name	Description	Milestones
Can't believe it worked!	Selenium tests passed	5, 10, 20
Bug Finder	Selenium tests fixed after not passing	1, 5, 10
Inspector Gadget	WebElements found using any locator strategy	10, 25, 50
Inspector Gadget: ID Expert	WebElements found using the 'ID' locator strategy	3, 5, 10
Inspector Gadget: Name Expert	WebElements found using the 'Name' locator strategy	3, 5, 10
Inspector Gadget: CSS Expert	WebElements found using the 'CSS Selector' locator strategy	3, 5, 10
Inspector Gadget: XPath Expert	WebElements found using the 'XPath' locator strategy	5, 15, 30
The Web Surfer	Websites visited	5, 10, 15
Back to...	WebDriver's back method executed	5
...the Future	WebDriver's forward method executed	5
F5 master	WebDriver's refresh method executed	5
Click addicted	Click method called on WebElements	5, 15, 30
Keyboard Master	Send_keys method called on WebElements	3, 5, 10
Display Detective	IsDisplayed method called on WebElements	3, 5, 10
Checkmark Champion	IsSelected method called on WebElements	3, 5, 10
The Validator	IsEnabled method called on WebElements	3, 5, 10
Textual Treasure Hunter	GetText method called on WebElements	5, 15, 30
Attribute Archaeologist	GetAttribute method called on WebElements	5, 10, 15
Styling Virtuoso	GetCssValue method called on WebElements	3, 5, 10
Keyblade Wielder	Attempted logins	1
Form Filler	Submit method called or submit buttons clicked	3, 5, 10
Title Tracker	WebDriver's getTitle method executed	5, 10, 15
Click-and-Dismiss	Interacting with an alert	1, 3, 5

Table 3.4: Project Achievements

Name	Description	Milestones
Can't believe it worked!	Selenium tests passed	10, 50, 100
Bug Finder	Selenium tests fixed after not passing	5, 25, 50
Inspector Gadget	WebElements found using any locator strategy	25, 50, 100
The Web Surfer	Websites visited	10, 25, 50
Click Addicted	Click method called on WebElements	15, 30, 50
Keyboard Master	Send_keys method called on WebElements	5, 15, 30
Textual Treasure Hunter	GetText method called on WebElements	15, 30, 50
Keyblade Wielder	Attempted logins	3, 5, 10
Click-and-Dismiss	Interaction with an alert	3, 5, 10
Suit Up	Edit your profile information for the first time	1
Showcase Legend	Edit your achievement showcase for the first time	1
Baby steps	Run your first Selenium test	1
Buggy beginnings	Fail your first Selenium test	1
Morning Person	Execute a new test between 6 a.m. and 8 a.m.	5, 10, 15
Night Owl	Execute a new test between 11 p.m. and 3 a.m.	5, 10, 15

Table 3.5: Global Achievements

Name	Description	Milestones
Inspector Gadget	WebElements found using any locator strategy	5
The Web Surfer	Websites visited	3
Click Addicted	Click method called on WebElements	5
Morning Person	Execute a test between 6 a.m. and 8 a.m.	1
Night Owl	Execute a test between 11 p.m. and 3 a.m.	1
Can't believe it worked!	Selenium tests passed	3
Keyboard Master	Send_keys method called on WebElements	3

Table 3.6: Daily Tasks

Chapter 4

Tool Validation

After the development phase of the gamification plugin and the gamification library, an experiment with a sample of four people was conducted to evaluate the tool's characteristics.

4.1 Sample of a Testing Session

Before subjecting the developed tool to external users, a test suite was designed to assess its performance and behavior. In particular, ensuring the tool had as few flaws as possible was crucial in creating a smooth experience for its users.

The validation test suite of the tool was performed on the JetBrains website[1] and comprises three tests. These tests are automatically created by IntelliJ IDEA when a user creates a Selenium project, to provide an example of its use to the developers. The operations performed in the tests can be summarized as follows:

- **Test 1**: In the website, the test looks for the search bar. After finding the element, it inputs the string "Selenium" and clicks the search button. On the new page, it checks that the "Selenium" string is present in the new search bar. The list of events generated by this test can be seen in Listing A.1.
- **Test 2**: In the website, the "Developer Tools" button is clicked. Afterwards, the script checks that the popup menu appeared.
- **Test 3**: In the website, the "Developer Tools" button is clicked. In the new popup menu, the "Find your tool" element is clicked. It then checks that the product list is displayed and that the title of the webpage is correct.

[1] `https://www.jetbrains.com/`

The tests ran without any particular issues and all three succeeded. The achievement progress these tests generated can be seen in Table 4.1. For clarity reasons, the achievements that did not gain any progress are not mentioned.

Name	**Progress**
Global Achievements	
Can't believe it worked!	3
Inspector Gadget	9
The Web Surfer	2
Click Addicted	4
Keyboard Master	1
Baby Steps	1
Project Achievements	
Can't believe it worked!	3
Inspector Gadget	9
Inspector Gadget: ID Expert	1
Inspector Gadget: CSS Expert	5
Inspector Gadget: XPath Expert	3
The Web Surfer	2
Keyboard Master	1
Display Detective	2
Attribute Archaeologist	1
Title Tracker	1

Table 4.1: Achievements Progress after Sample Testing Session

The only achievements that reached a milestone are "Baby Steps", which is marked as completed, and "Inspector Gadget: CSS Expert", which reached the third milestone. Therefore, this test suite awarded the user some points. The final level registered in this profile is **Level 1**, with **40 / 50** experience points. It is also important to note that the daily task that was generated during this session could not be completed, because it required running the tests early in the morning. Additionally, the achievements regarding the edit of the user profile and user showcase were not considered during these tests: the goal of the suite was only to verify that the Selenium WebDriver's operations were correctly tracked.

After checking that the tool did not have any particular flaws, an experiment, that was conducted to evaluate the developed plugin, will be described in the following section.

4.2 Experiment

4.2.1 Participant Demographics

Using the gamification plugin and the gamification library requires that the participants have some programming knowledge in their background. Since the goal of the tool is to incentivize GUI testing, the users have been selected among people who study or work in the Information Technology (IT) field. The four selected participants have all a master's degree qualification obtained at *Politecnico di Torino* in computer engineering. Two of them are now employed, and the other two are pursuing a PhD at *Politecnico di Torino*.

Some additional details were gathered from the first section of the administered survey to the participants, as can be seen in Table 4.2.

ID	Question (Type)
1.1	Age range (Multiple Choice)
1.2	Occupation (Multiple Choice)
1.3	How many years of experience do you have in object-oriented programming? (Open)
1.4	How many years of experience do you have in web application programming? (Open)
1.5	Do you have previous experience in web application testing? (Multiple Choice)
1.6	Which tools did you use for testing web applications? (Open)
1.7	How familiar are you with the IntelliJ IDEA IDE? (Likert)

Table 4.2: Survey Questions: Part one

The results showed that all the participants are between 25 and 30 years old. This is an important metric to consider because people below the age of 50 years old are more inclined to understand some mechanics regarding the tech gaming world [41] compared to the other age range.

All the participants had at least 2 years of experience in Object-Oriented Programming (OOP) and at least 1 year of experience in web application programming. This fact does not come as a surprise, since the users selected for this experiment have all a master's degree in computer engineering. It is more interesting, since it is the goal of the proposed tool, to analyze the experience the participants have in web application testing. As seen in Figure 4.1, the majority of participants had previous experience in web application testing, but only one among them had previously used Selenium. Other frameworks that were mentioned by the participants are *Jest*, *Cypress*, *Scout*, and *SikuliX*.

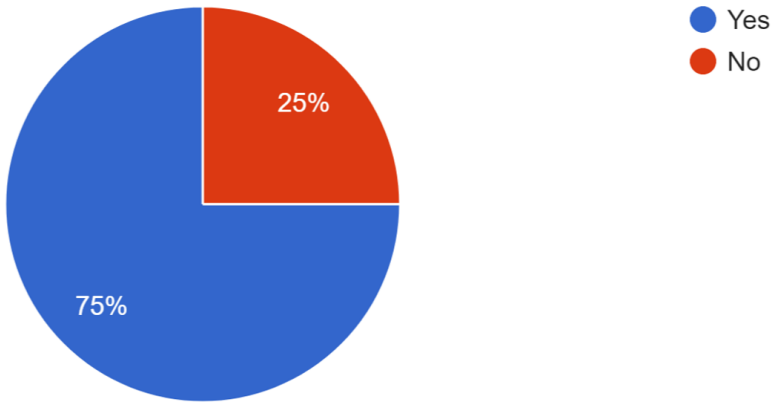

Figure 4.1: Q1.5 Web Application Testing Experience

Even if three out of four participants never used Selenium WebDriver, their knowledge about OOP and web applications programming helped them to quickly grasp its use and to successfully perform the assigned tasks during the experiment.

The final question about their background is aimed at investigating the participants' familiarity with the IntelliJ IDEA IDE. The answers to this question can be seen in Figure 4.2. This background question is important because it helps to

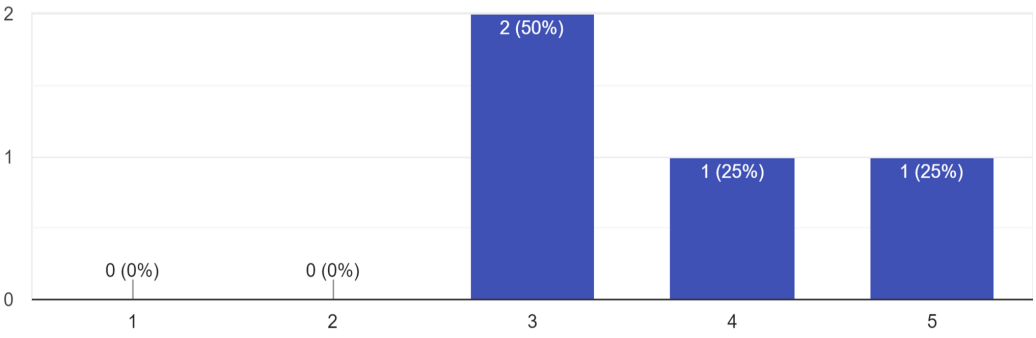

Figure 4.2: Q1.7 IntelliJ IDEA IDE Familiarity

contextualize the feedback given by the users. Understanding how the plugin is perceived by people with less familiarity with the IDE and by people who are more accustomed to it is crucial to determine its success. The answers revealed that all the users had a certain level of familiarity with the IntelliJ IDEA IDE, or at least heard about it during their careers.

4.2.2 Methodology

The conducted experiment was composed of three tasks the user had to complete in a maximum of 45 minutes. Before the execution of the tasks, a small introduction to GUI testing and, in particular, to Selenium WebDriver was made to the participants unfamiliar with this framework. The purpose of this introduction was to put all the users on the same base level so that they would be able to complete the assigned tasks. After the session, a survey was administered to the participants to gather feedback and evaluate the developed tool's characteristics.

The aim of the three assigned tasks was performing GUI testing on a popular e-commerce website, *Amazon*[2], to investigate the use of the developed plugin by the participants. The purpose of the experiment is to analyze how the users interact with the plugin and if they change their approach to GUI testing by following the introduced gamification elements. For these reasons, the three tasks had a clear goal but with room for maneuver for the participants:

- **Task 1**: The goal of this task was to search for a specific product on the webpage, click on the first result, and check that its price is lower than a defined threshold.
- **Task 2**: Starting from the product page found in task 1, the goal of this task was to go to the reviews page and check the correctness of the review's title and author of both top positive and top critical review.
- **Task 3**: Starting from the product page found in task 1, the goal of this task was to add the product to the cart, go to the cart page, verify the information about the product (name, quantity, price, subtotal) and, finally, go to the checkout page and check if the button to create a new account is present.

During the testing session, the users were able to see Selenium code examples online, in other projects, and also free to check the online documentation if they needed to. Additionally, they could exploit the *Test Automation* plugin offered by IntelliJ IDEA Ultimate. This plugin, as cited in the previous chapter, has an embedded web inspector that can generate XPath or CSS locators of the clicked web element. Finding web elements, even for the participants who have never used Selenium, becomes much easier. The interface of the plugin can be seen in Figure 4.3.

Finally, before letting the participants do the assigned tasks and after introducing them to the Test Automation plugin, the users were allowed to explore and interact with the gamification plugin to get accustomed to it.

[2] `https://www.amazon.com/`

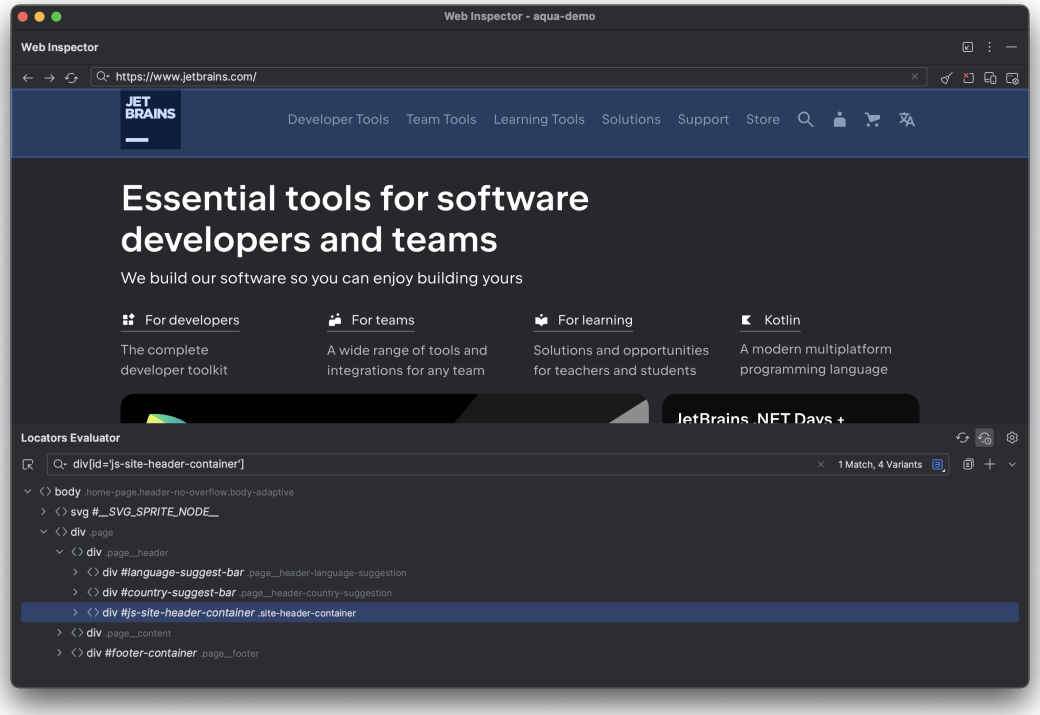

Figure 4.3: Test Automation Plugin Web Inspector

4.2.3 Execution Environment

The experiment followed the methodology previously defined. The participants carried out the testing sessions on a project on the same machine with the following characteristics:

- CPU: AMD Ryzen 7 5800H
- Operating System: Windows 10 Pro
- IDE: IntelliJ IDEA 2023.2.2 (Ultimate Edition)
  Build: #IU-232.9921.47
- Selenium version 4.15.0
- JUnit version 5.9.3
- Gradle version 8.3
- Java SE version 20.0.2

Since reproducing the testing scenario needed the installation and configuration of several tools, it has been preferred to let the users perform the experiment on the same machine. Had this solution not been implemented, participants would have been required to configure the following tools: install IntelliJ IDEA (Ultimate Edition), configure a Selenium project, install the Test Automation plugin, install the gamification plugin, and, finally, import the gamification library in their project. Moreover, the Test Automation plugin can only be installed on commercial JetBrains IDEs, which require a paid license.

It is worth mentioning that the sessions were conducted on a laptop with the US International keyboard layout. Since the participants were not accustomed to this type of layout and its key combinations, overall they took a longer time to perform the assigned tasks than what would have required them on their personal computers.

Finally, the project the users had to implement their tasks was already set up. The gamification plugin was installed, and the gamification library was already imported via Gradle. Additionally, the "BeforeEach" method that creates the Selenium WebDriver, attaches the gamified listener, and then opens the webpage was already present, together with the "AfterEach" method that quits the WebDriver. Before the session, the participants were made aware of what these methods do, how the library was imported, and how it is utilized in the code.

4.2.4 Results

For the most part, the four testing sessions ran without any particular issue that is worth mentioning. In the last two testing sessions, a small nuisance appeared that was not expected: the site the testing was performed on started asking for a CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart) each time the Selenium WebDriver was opened. A temporary fix was making the WebDriver wait some seconds when opening the page so that the participant could resolve the CAPTCHA manually. While this slightly slowed down the testing process and caused some minor user frustration, all things considered, the impact was negligible.

After each session, the achievements statistics of the user were registered and are shown all together in Table 4.3. For the sake of clarity, the table includes only the achievements for which at least one user has made some progress. The majority of participants managed to complete all three tasks in the scheduled time. Only one of them did not succeed because they had some issues finding the elements on the webpage, and they appeared to prioritize testing aspects unrelated to the tasks' primary objectives. The other testing sessions did not present any issues to mention.

An in-depth analysis of the participants' achievements statistics will now follow.

Name	User 1	User 2	User 3	User 4
Global Achievements				
Can't believe it worked!	2	3	3	3
Bug Finder	3	3	3	2
Inspector Gadget	10	17	14	13
The Web Surfer	3	6	6	5
Click Addicted	2	6	3	7
Keyboard Master	1	1	-	1
Textual Treasure Hunter	7	9	10	8
Suit Up	1	1	1	1
Showcase Legend	1	1	1	1
Baby Steps	1	1	1	1
Buggy Beginnings	1	1	1	1
Project Achievements				
Can't believe it worked!	2	3	3	3
Bug Finder	3	3	3	2
Inspector Gadget	10	17	14	13
Inspector Gadget: ID Expert	-	-	2	2
Inspector Gadget: CSS Expert	1	3	-	9
Inspector Gadget: XPath Expert	7	14	12	2
The Web Surfer	3	6	6	5
Back to...	-	-	-	1
...the Future	-	-	-	1
Click Addicted	2	6	3	7
Keyboard Master	1	1	-	1
Display Detective	-	1	-	-
Textual Treasure Hunter	7	9	10	8
Attribute Archaeologist	-	-	1	-
Form Filler	-	2	2	2

Table 4.3: Achievements Progress after Experiment

One thing that stands out is that most of the web elements were found using as locators strategies XPath or CSS, with a prevalence of XPath. This is not surprising because the Test Automation plugin, which the users were free to use, generates an XPath as the first choice. Even though the web element owns the id property, the plugin will create an XPath or CSS locator that incorporates the id. Only some of the users went beyond the tasks' goals and tried to gain progress in other achievements. User 4 is the only one who tried to use the Selenium WebDriver to

navigate back and forth, gaining progress respectively in "Back to..." and "...the Future" achievements. User 3 is the only one who checked the presence of the create account button by using the displayed method and is the only one who tried to get the attribute of a web element. Additionally, User 3 did not fully grasp what their goal was in the first task. They were the only user who did not perform any form of web application testing previously, therefore this could be one of the reasons. Instead of searching for the product on the site by using an automated browser with Selenium WebDriver, they navigated immediately to the product page, which they searched manually on the browser. This is the reason they have fewer sites visited, fewer clicks, and did not make any progress in the "Keyboard Master" task. The other achievements do not need a detailed analysis: "Can't believe it worked!" counts the number of tasks the users were able to complete, and "Bug Finder" shows the number of tests the user successfully addressed after they initially failed. Other metrics, like "Click Addicted" and "Textual Treasure Hunter" reflect the steps the participants had to do to navigate between sites or to check the correctness of the presented information.

The progress in the various achievements has also led the user profiles to level up. A comparative analysis between their levels and experience points can be seen in Table 3.1. These results are reasonable and give credit to the previous

User	Level	EXP
User 1	2	50
User 2	2	80
User 3	2	60
User 4	2	80

Table 4.4: User Levels after Experiment

analysis. User 1 is the participant who made less progress: it is not surprising since is the only one who did not manage to complete all the tasks. Moreover, User 3 is also a step behind User 2 and User 4. This can be explained by the previously mentioned fact that User 3 did not fully grasp what the goal of the first task was, therefore they did not complete it in the way it was expected. The final fact, that is important to mention and to take into consideration, is that all participants were assigned a daily task that could be completed during the experiment. Hence, no user was at a disadvantage for the unpredictability of the daily task.

The final achievements that are worth analyzing are "Suit Up" and "Showcase Legends". These awards demonstrate that all the participants interacted with the gamification elements introduced by the plugin, in particular with the possibility of editing their profile page and their trophy showcase. Figure 4.4 shows how the users decided to edit their profile. The only difference is that the displayed names

were edited to respect the participants' privacy. Most participants decided to use

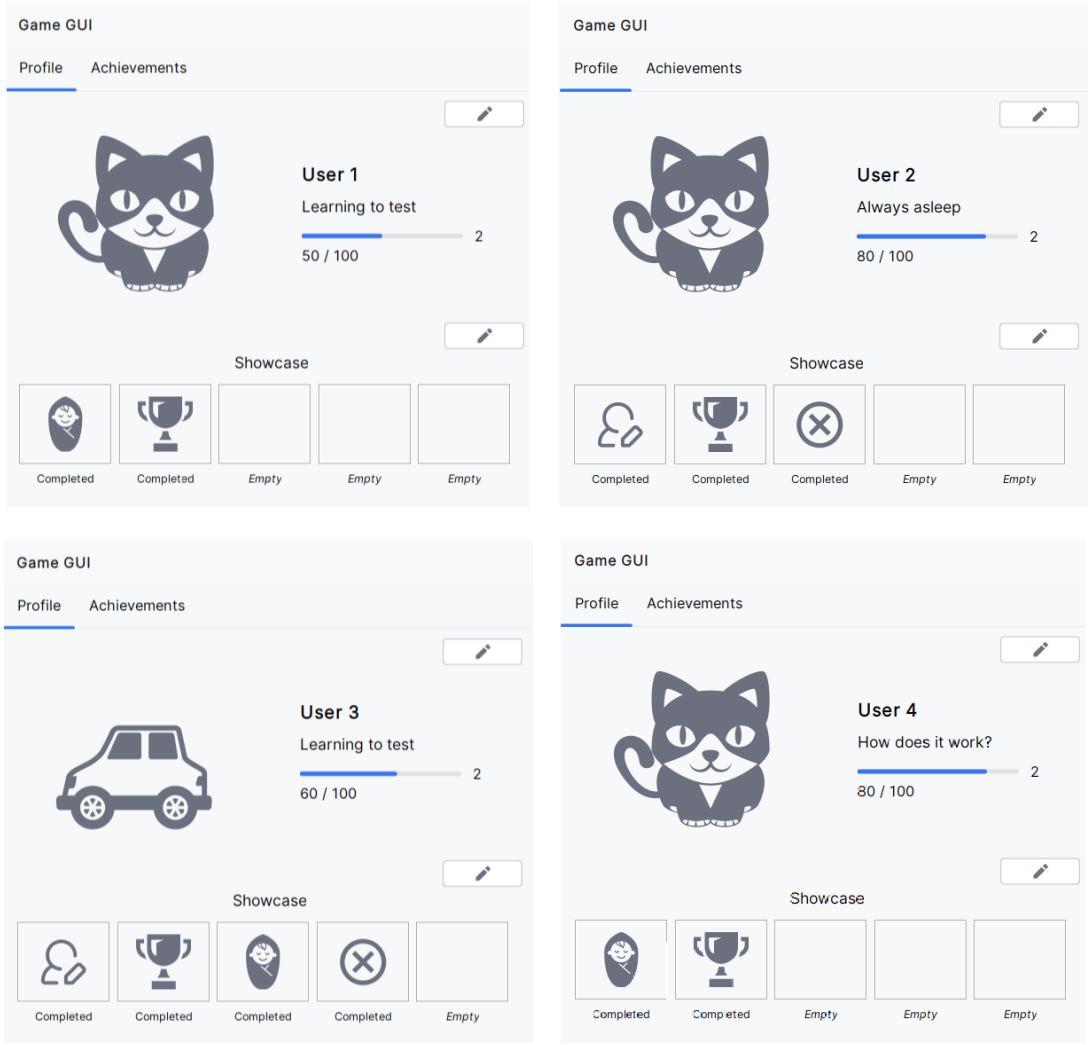

Figure 4.4: Users Profile Pages

the cat image as their profile picture. Both the car and the cat picture are unlocked at Level 2, along with a dog image that was not chosen by anyone. The users did not put all the unlocked global achievements in their trophy showcase: most of them decided to put only the ones they were most proud of or the ones with the nicer icon representing them.

Usability

The second part of the survey that was administered to the participants after the testing session can be seen in Table 4.5. It is a standard System Usability Scale[42]

(SUS) questionnaire, whose purpose is to measure in a "quick and dirty" fashion the usability of a system. These SUS questions, as also explained to the participants,

ID	Question (Type)
2.1	I think I would like to use this system frequently (Likert)
2.2	I found the system unnecessarily complex (Likert)
2.3	I thought the system was easy to use (Likert)
2.4	I think I would need the help of a technical person to be able to use this system (Likert)
2.5	I found the various functions in this system well-integrated (Likert)
2.6	I thought there was too much inconsistency in this system (Likert)
2.7	I would imagine that most people would learn to use this system very quickly (Likert)
2.8	I found the system very cumbersome to use (Likert)
2.9	I felt very confident using the system (Likert)
2.10	I needed to learn a lot of things before I could get going with this system (Likert)

Table 4.5: Survey Questions: Part two

are referred only to the gamification plugin developed and its tool window. The users are not requested to evaluate the usability of the IntelliJ IDEA IDE or of other aspects they entered into contact with. The answers to part two of the survey are shown in Figure 4.5. The results are generally positive: there are some questions where the answers are neutral but never negative towards the system. The final SUS score for the system, calculated following the SUS conventions, is **93.75**. Based on research, a score above 68 would be considered above average, but not many considerations can be made apart from that. However, it was somewhat expected the score to be high. The participants needed to evaluate a simple window with few interfaces, hence the margin for errors was not very wide. Nevertheless, having a good and usable user interface is a positive outcome. One of the main benefits of gamification is improving the user experience[2] and a well-crafted contributes to this goal.

Gamification Elements

The third part of the administered survey includes questions about the gamification elements the participants interacted with during the testing session, as can be seen in Table 4.6. The answers to the Likert questions from 3.1 through 3.8 can be

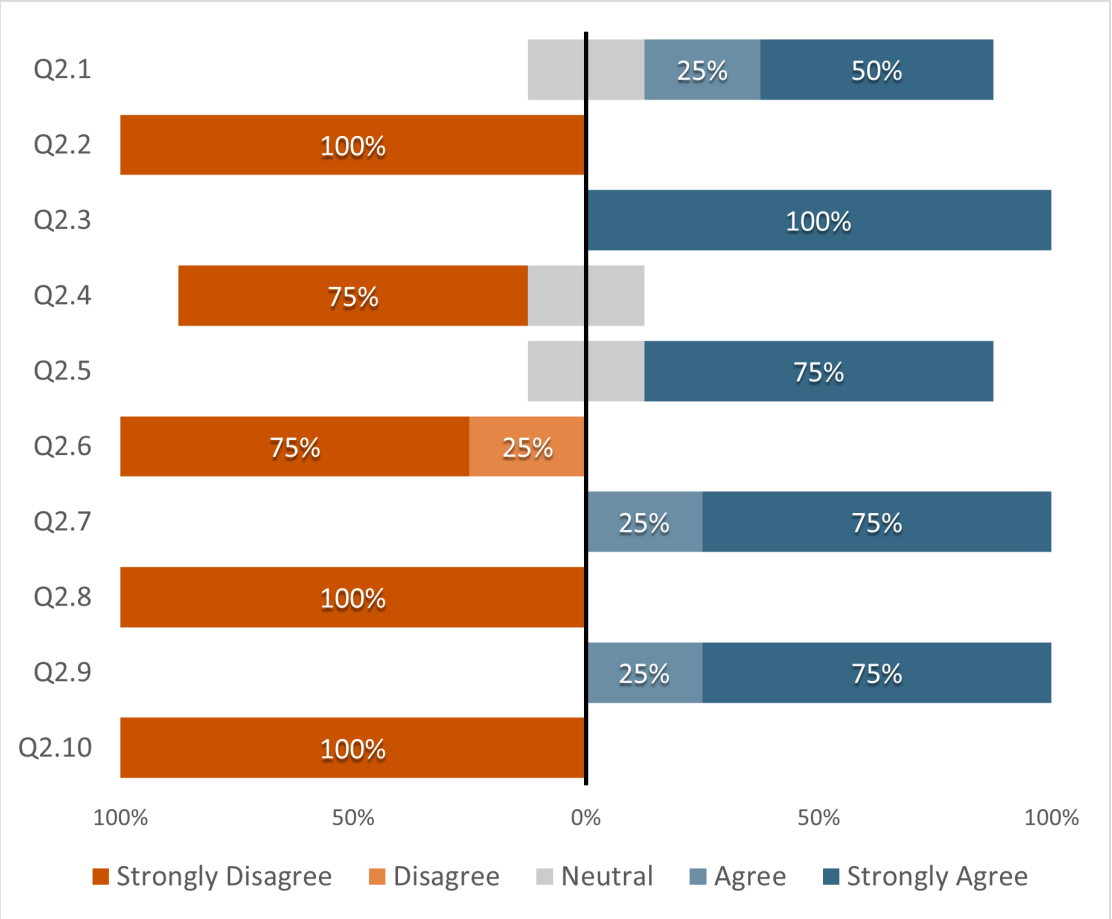

Figure 4.5: System Usability Scale Results

seen in Figure 4.6. The first question covers the general feelings of the participants towards the gamification elements implemented in the system and how the tool worked as a whole. In general, the users did not have any doubts about the system: all the mechanics were clear. The reason is also that the participants are young and are used to interacting with game elements in other contexts almost every day. The following questions focus on asking about each gamification element they encountered during the testing session.

The sentiment about project achievements is positive, meanwhile the feeling about global achievements tends to be more neutral. The users perceived better the project achievements because they had lower milestones, hence they could be achieved more easily. Regarding global achievements, one of the opinions expressed is that they are too similar to the project ones and they hardly noticed the difference between the two types. Having some of the same achievements with

ID	Question (Type)
3.1	The tool functionings and mechanics were clear to me (Likert)
3.2	How did you perceive the presence of project achievements in encouraging more in-depth testing? (Likert)
3.3	How did you perceive the presence of global achievements in encouraging more in-depth testing? (Likert)
3.4	How did you perceive the presence of the achievement showcase in encouraging the completion of global achievements? (Likert)
3.5	How did you perceive the presence of a daily task in encouraging daily testing? (Likert)
3.6	How did you perceive the presence of a progress bar and a user level in pushing you to complete additional achievements? (Likert)
3.7	How did you perceive the presence of unlockable titles and icons in pushing you to complete additional achievements? (Likert)
3.8	How did you perceive the presence of notifications in keeping you updated on your progress? (Likert)
3.9	Do you believe the used tool can be easily integrated into an existing testing environment (working or studying)? (Likert)
3.10	Which of the gamification elements did you find most useful? (Multiple Choice)
3.11	What were the main issues you encountered during the testing session? (Open)
3.12	Have you got any suggestions, comments, or unsolved doubts on the presented tool? (Open)

Table 4.6: Survey Questions: Part three

higher milestones is a good solution, but more variety is needed. Q3.5 asked the participants about daily tasks. The users had good feelings about them and thought that these tasks could stimulate them to perform GUI testing more regularly.

The achievement showcase, one of the other introduced gamification elements, also had a generally positive sentiment. The participants thought it was a good way to express themselves and show their accomplishments. Additionally, some of them expressed the opinion that they wished the showcase, and their profile page, were public or shareable with other users to "boast" with their colleagues. It seems that in this answer they considered the potentiality that this gamification element could have more than the actual presented implementation.

The progress bar and user level were also well received, even though, as in global

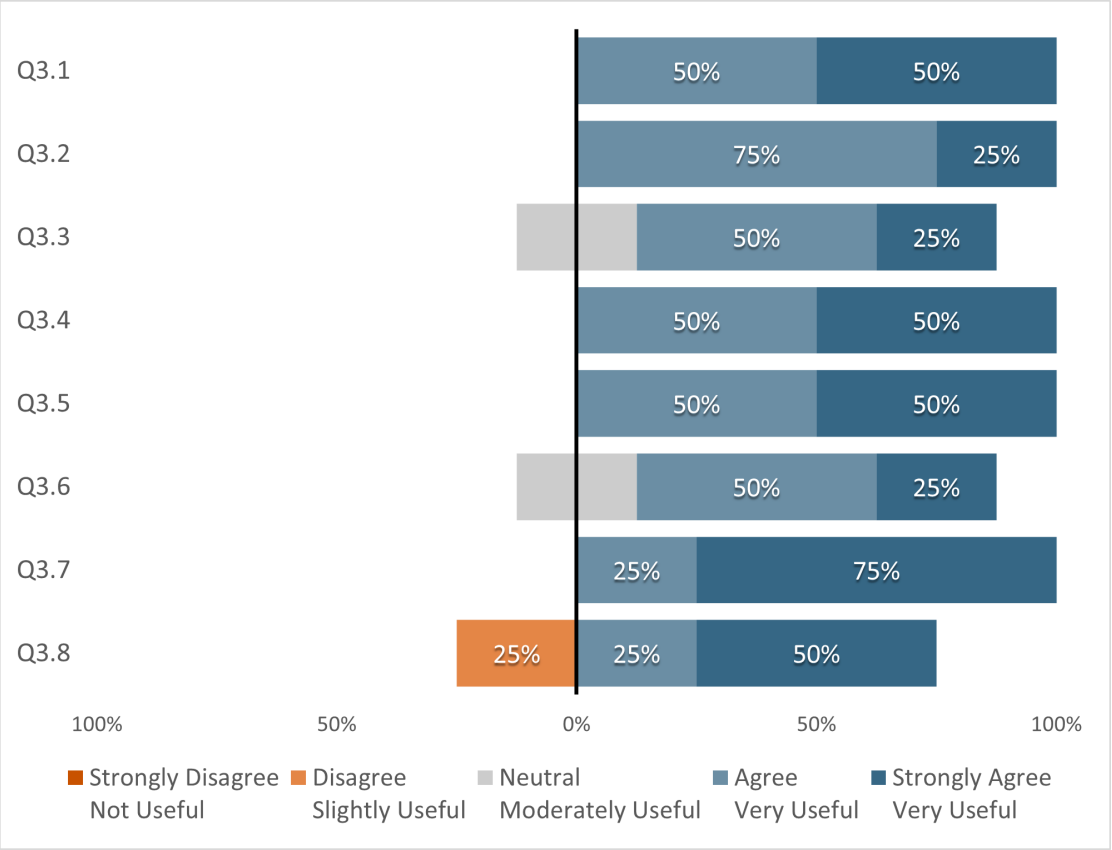

Figure 4.6: Gamification Elements Survey Results

achievements, there were some neutral opinions about them. Having only a number that grows when you complete achievements seemed to be generally a weaker drive to perform GUI testing and complete achievements.

In Q3.7, the participants were asked about the unlockables present in the tool: user icons and titles. As will be shown afterward, this was generally the best-received introduced element by this sample of users. The fact that the unlockables are strictly connected to the user level, made it necessary to ask about the two features separately. It seems that the user level alone was not sufficient to demonstrate their expertise and use of the gamification plugin with other people. Instead, having something more meaningful to aspire to, like icons and titles, seems to be a stronger drive to complete achievements.

The last plugin element the users were asked to express their feelings for was notifications. Even if some users expressed a positive feeling, the majority of them seemed to agree on the fact that they were hard to notice sometimes. The implemented notifications are the standard provided by the IntelliJ IDEA IDE,

hence not much can be done on this front. The tool implements notifications that disappear after a certain amount of time to not distract the users too much: this can be the reason why some of the participants missed them. After telling the users there were notifications sent by the plugin, they immediately started paying more attention. Probably they are used to dismiss or ignore the notifications sent by the IDE. A solution to this problem could contemplate the use of other ways to alert the users of their advancements. A possible idea could be implementing some notifications directly into the tool window of the plugin, or highlighting the achievements that recently received some progress.

Finally, question Q3.10 asked the participants which were their favorite gamification elements and the results can be seen in Figure 4.7. As can be seen from

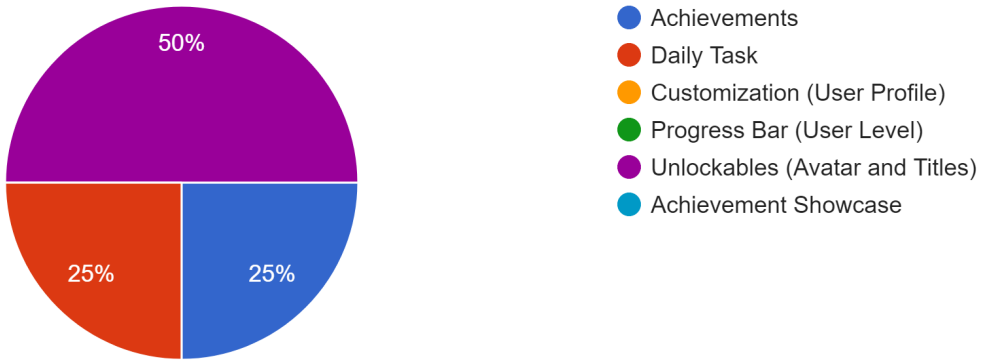

Figure 4.7: Q3.10 Preferred Gamification Element

the graph, the participants preferred the unlockables, the daily tasks, and the achievements among all the gamification elements. After analyzing the previous answers, this outcome is not surprising. The unlockables played a great role in driving the participants to complete achievements and perform GUI testing. Some of the users see them as the biggest motivator: they are content they have unlocked something through their efforts. Other participants were satisfied by seeing their progress on the achievements and did not feel like they needed additional rewards. The daily tasks were also well received: they are thought to be a good incentive to test regularly and make the users curious to see which task will appear every day.

Final Considerations

There are still three questions whose answers need to be analyzed: Q3.9, Q3.11, and Q3.12. The first one asked the participants if they thought the system was easy to integrate into existing testing environments and the answers can be seen in

Figure 4.8. Before asking this question, the users were made aware of all the steps

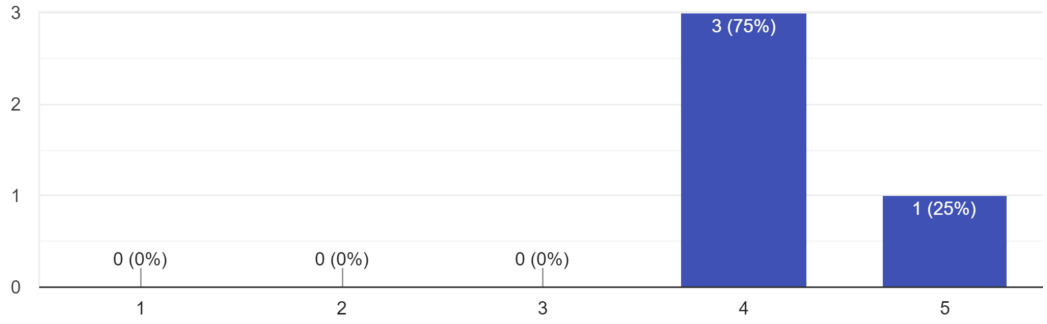

Figure 4.8: Q3.9 Ease of Tool Integration

that were followed to create the experiment environment that they were testing on. The participants still agreed to the fact that it was not too complex to do. However, these answers need to be re-evaluated in a future experiment where the participants need to integrate the gamification tool into their environment by themselves.

Questions 3.11 and 3.12 are open-ended questions where the participants were able to express freely the issues they encountered or other comments they wanted to make. List of the received opinions:

- (issue) "Find the right way to find an element on the page" and "I had no previous experience in Selenium": these problems do not regard in particular the proposed tool, but they are addressed to the Test Automation plugin and the lack of familiarity with Selenium WebDriver.
- (issue) "There were some issues with the site's CAPTCHA": this issue has already been discussed in a previous section, but it is still worth mentioning that the testing session of some participants was slightly annoying to them due to this fact.
- (suggestion) "Give the possibility to share the achievements/profile with friends" and "Enable a way to brag about your achievements with your coworkers": these are good suggestions that should be added in the future but with the current implementation of the tool it is not possible.
- (suggestion) "Give more emphasis to the rarest achievements": this suggestion refers to the fact that global achievements, even though they have higher milestones and are more difficult to complete, are treated like the project ones. The participant suggested introducing a way to make them stand out, like putting an animation to their icon, changing the color, or using a special notification.

In conclusion, the gamification elements were well received by all the participants of the sample group. They thought them to be strong motivators to perform GUI testing and liked their mechanics and functioning. Some of them expressed the opinion that they would use the gamification tool in their daily work if it was available.

Even though the participants' opinions were enthusiastic, there are still some doubts about the effects of the tool. By observing the users, it seemed that most of them were focused on finishing the assigned tasks and did not interact much with the plugin to see their progress or to follow specific achievements. This could be for several reasons: for instance, unfamiliarity with Selenium WebDriver and the machine they were testing on. A larger-scale experiment needs to be carried out to see if the tool increases the efficiency of the testers or if the gamification elements only increase the users' satisfaction. Additionally, it would be useful to understand which of the gamification elements is preferred and is a stronger drive to perform GUI testing. A comparison session between users who do not use the gamification plugin and users who do should be conducted in the future the assess the real potentialities of the developed tool.

Chapter 5

Conclusions

The conducted experiment showed promising results about the tool integration in scripted GUI testing environments. Despite that, the current tool limitations need to be analyzed to lay the foundation for future works.

5.1 Tool Limitations

The developed tool does not present any evident bug or flaw in its implementation. However, it does have certain limitations, which will be explained below.

- The current plugin needs the gamification library, imported by the user, to be able to track the Selenium WebDriver actions. Finding a way to incorporate the tracking directly in the plugin would drastically improve the user experience of the testers.

- The tracked actions of the Selenium WebDriver are very simple in the developed tool because they are dependent on the available methods offered by the WebDriverListener interface. If another way was found to track all the operations performed, more advanced achievements could be developed.

- The plugin relies on the IntelliJ Platform "PersistingStateComponent" interface to store data about the projects and the user profile. This solution is not ideal because users can externally modify or delete the XML files where this information is stored. Additionally, adding too much information to these files risks slowing down the entire plugin. In the context taken into consideration, this is an acceptable solution since the users perform small testing sessions and the plugin is not available worldwide in the JetBrains Marketplace. A possible resolution to this problem is the implementation of an external database. Through its use, a user leaderboard can be implemented directly in the plugin.

Additionally, a search feature can be added to look at the profiles of other users in the plugin.

5.2 Future Works

Even though the participants of the experiment were enthusiastic about the introduced gamification features, the actual effects of the developed tool still need to be assessed. A large-scale experiment should be conducted to compare users who do not use the gamification elements to users who use them and see if the plugin improves the testers' efficiency and not only their morale. Additionally, a large-scale experiment would better highlight which are the most useful gamification elements to stimulate GUI testing and which are not to the proposed goal. It would also be a good opportunity to test the tool on other users' machines and to finally see if its integration in an existing environment is straightforward or not.

The plugin's implementation could be improved by making some changes. Additional research on different methods to track the Selenium WebDriver could be done, to avoid using the gamification library and make the whole process more transparent to the user. Moreover, it could lead to the possibility of tracking more specific WebDriver actions that could not be done in the current iteration, being constrained by the WebDriverListener interface.

The developed tool implementation can also be expanded with additional features. Adding an external database to create a social system inside the plugin would stimulate the Social Influence core drive defined by the Octalysis framework. In this regard, an attempt has been made by using the achievements showcase as a gamification element but, as highlighted by the user's feedback, the absence of the possibility of sharing their own goals directly from the plugin made it less useful than expected.

Other gamification elements can also be introduced in the plugin. For instance, a narrative component, or quests that need to be completed in a particular order. As mentioned before, more social features would make a good addition but also the introduction of a virtual currency and a virtual shop where the user can buy virtual goods. More customization options for user profiles can also be added, or the presence of Easter eggs and hidden achievements when the user executes particular actions. These are only some suggestions but the final decision pertains to the people who will expand on the developed tool.

In wrapping up this section and the thesis work, these improvements and future explorations are aimed at improving the plugin's capabilities, ensuring adaptability, and contributing to the research of gamification solutions in software testing.

Appendix A

Additional Resources

Listing A.1: JSON Event List Example

```
[
    {
        "id": "",
        "url": "www.jetbrains.com/",
        "eventType": "NAVIGATION"
    },
    {
        "id": "[data-test='site-header-search-action']",
        "url": "www.jetbrains.com/",
        "eventType": "LOCATOR_CSS"
    },
    {
        "id":
    ↪ "/html[1]/body[1]/div[1]/div[1]/div[3]/header[1]
    ↪ /div[1]/div[1]/div[2]/div[1]/div[1]/div[1]/div[1]
      /div[1]/button[1]",
        "url": "www.jetbrains.com/",
        "eventType": "ELEMENT_CLICK"
    },
    {
        "id": "[data-test='search-input']",
        "url": "www.jetbrains.com/",
        "eventType": "LOCATOR_CSS"
    },
    {
        "id":
    ↪ "/html[1]/body[1]/div[1]/div[1]/div[3]/header[1]
    ↪ /div[1]/div[1]/div[2]/div[4]/div[1]/div[1]/div[1]
    ↪ /label[1]/div[1]/div[1]/input[1]",
```

```
25          "url": "www.jetbrains.com/",
26          "eventType": "ELEMENT_SEND_KEYS"
27      },
28      {
29          "id": "button[data-test='full-search-button']",
30          "url": "www.jetbrains.com/",
31          "eventType": "LOCATOR_CSS"
32      },
33      {
34          "id":
    ↪ "/html[1]/body[1]/div[1]/div[1]/div[3]/header[1]
    ↪ /div[1]/div[1]/div[2]/div[4]/div[1]/div[1]/div[2]
    ↪ /div[1]/button[1]",
35          "url": "www.jetbrains.com/",
36          "eventType": "ELEMENT_CLICK"
37      },
38      {
39          "id": "input[data-test='search-input']",
40          "url": "www.jetbrains.com/",
41          "eventType": "LOCATOR_CSS"
42      },
43      {
44          "id":
    ↪ "/html[1]/body[1]/div[1]/div[1]/div[3]/header[1]
    ↪ /div[1]/div[1]/div[2]/div[4]/div[1]/div[1]/div[1]
    ↪ /div[1]/div[1]/label[1]/div[1]/div[1]/input[1]",
45          "url": "www.jetbrains.com/",
46          "eventType": "ELEMENT_CLICK"
47      },
48      {
49          "id":
    ↪ "/html[1]/body[1]/div[1]/div[1]/div[3]/header[1]
    ↪ /div[1]/div[1]/div[2]/div[4]/div[1]/div[1]/div[1]
    ↪ /div[1]/div[1]/label[1]/div[1]/div[1]/input[1]",
50          "url": "www.jetbrains.com/",
51          "eventType": "ELEMENT_ATTRIBUTE"
52      }
53  ]
```

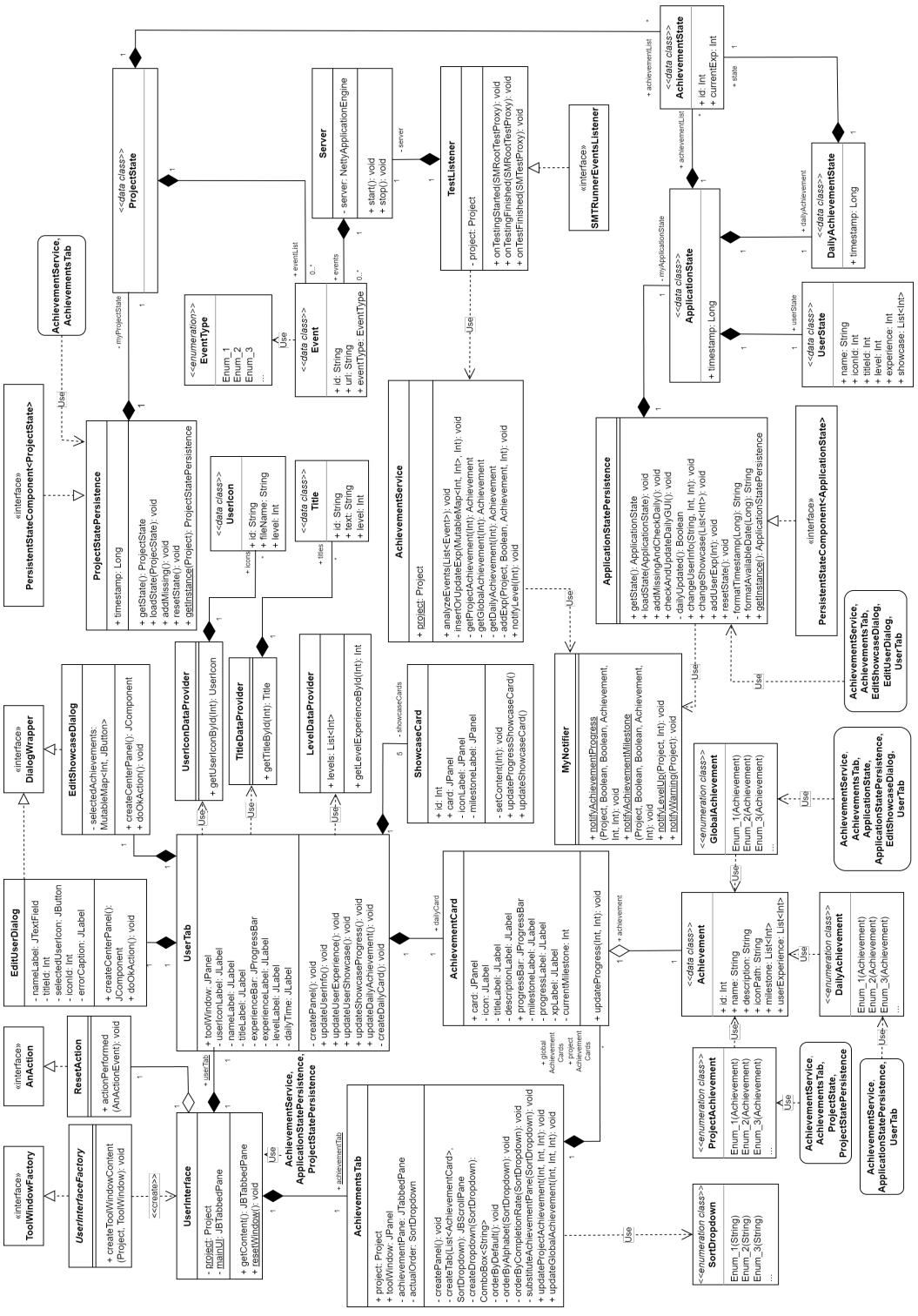

Figure A.1: Gamification Plugin UML Class Diagram

Bibliography

[1] Mary Jean Harrold. «Testing: A Roadmap». In: *Proceedings of the Conference on The Future of Software Engineering*. ICSE '00. Limerick, Ireland: Association for Computing Machinery, 2000, pp. 61–72. ISBN: 1581132530. DOI: 10.1145/336512.336532. URL: https://doi.org/10.1145/336512.336532 (cit. on p. 3).

[2] Tommaso Fulcini, Riccardo Coppola, Luca Ardito, and Marco Torchiano. «A Review on Tools, Mechanics, Benefits, and Challenges of Gamified Software Testing». In: *ACM Comput. Surv.* 55.14s (July 2023). ISSN: 0360-0300. DOI: 10.1145/3582273. URL: https://doi.org/10.1145/3582273 (cit. on pp. 3, 10, 11, 18, 68).

[3] Mike Cohn. *Succeeding with Agile: Software Development Using Scrum*. 1st. Addison-Wesley Professional, 2009. ISBN: 0321579364 (cit. on p. 4).

[4] Michael Ellims, James Bridges, and Darrel C. Ince. «The Economics of Unit Testing». In: *Empirical Software Engineering* 11 (2006), pp. 5–31. URL: https://api.semanticscholar.org/CorpusID:6003929 (cit. on p. 4).

[5] Mario Linares-Vásquez, Kevin Moran, and Denys Poshyvanyk. «Continuous, Evolutionary and Large-Scale: A New Perspective for Automated Mobile App Testing». In: *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 2017, pp. 399–410. DOI: 10.1109/ICSME.2017.27 (cit. on p. 4).

[6] *STLC (Software Testing Life Cycle)*. URL: https://www.guru99.com/software-testing-life-cycle.html (visited on 11/23/2023) (cit. on p. 4).

[7] Vahid Garousi, Ali Mesbah, Aysu Betin-Can, and Shabnam Mirshokraie. «A systematic mapping study of web application testing». In: *Information and Software Technology* 55 (Aug. 2013), pp. 1374–1396. DOI: 10.1016/j.infsof.2013.02.006 (cit. on p. 6).

[8] Emil Borjesson and Robert Feldt. «Automated System Testing Using Visual GUI Testing Tools: A Comparative Study in Industry». In: *Proceedings - IEEE 5th International Conference on Software Testing, Verification and Validation, ICST 2012* (Apr. 2012). DOI: 10.1109/ICST.2012.115 (cit. on p. 7).

[9] LaShanda Dukes, Xiaohong Yuan, and Francis Akowuah. «A case study on web application security testing with tools and manual testing». In: *2013 Proceedings of IEEE Southeastcon*. 2013, pp. 1–6. DOI: 10.1109/SECON.2013.6567420 (cit. on p. 7).

[10] Maurizio Leotta, Diego Clerissi, Filippo Ricca, and Paolo Tonella. «Capture-replay vs. programmable web testing: An empirical assessment during test case evolution». In: *2013 20th Working Conference on Reverse Engineering (WCRE)*. 2013, pp. 272–281. DOI: 10.1109/WCRE.2013.6671302 (cit. on p. 7).

[11] Michel Nass, Emil Alegroth, and Robert Feldt. «Augmented Testing: Industry Feedback To Shape a New Testing Technology». In: Apr. 2019. DOI: 10.1109/ICSTW.2019.00048 (cit. on p. 8).

[12] Michel Nass, Emil Alegroth, and Robert Feldt. «On the Industrial Applicability of Augmented Testing: An Empirical Study». In: Aug. 2020. DOI: 10.1109/ICSTW50294.2020.00065 (cit. on p. 9).

[13] Sebastian Deterding, Dan Dixon, Rilla Khaled, and Lennart Nacke. «From Game Design Elements to Gamefulness: Defining "Gamification"». In: *Proceedings of the 15th International Academic MindTrek Conference: Envisioning Future Media Environments*. MindTrek '11. Tampere, Finland: Association for Computing Machinery, 2011, pp. 9–15. ISBN: 9781450308168. DOI: 10.1145/2181037.2181040. URL: https://doi.org/10.1145/2181037.2181040 (cit. on p. 10).

[14] Riccardo Gabellone. «Sviluppo di un tool di mutant injection nel contesto di un ambiente di GUI Testing con gamification». MA thesis. Politecnico di Torino, 2022. URL: https://webthesis.biblio.polito.it/22684/ (cit. on p. 10).

[15] Carlos Futino Barreto and César França. «Gamification in Software Engineering: A literature Review». In: *2021 IEEE/ACM 13th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*. 2021, pp. 105–108. DOI: 10.1109/CHASE52884.2021.00020 (cit. on p. 10).

[16] Karen Robson, Kirk Plangger, Jan H. Kietzmann, Ian McCarthy, and Leyland Pitt. «Is it all a game? Understanding the principles of gamification». In: *Business Horizons* 58.4 (2015), pp. 411–420. ISSN: 0007-6813. DOI: https://doi.org/10.1016/j.bushor.2015.03.006. URL: https://www.sciencedirect.com/science/article/pii/S000768131500035X (cit. on p. 10).

[17] Yu-kai Chou. *Actionable Gamification: Beyond Points, Badges, and Leaderboards*. Packt Publishing, 2019. ISBN: 9781839210778. URL: https://books.google.it/books?id=9ZfBDwAAQBAJ (cit. on pp. 11, 14).

[18] Yu-kai Chou. *The Octalysis Framework for Gamification & Behavioral Design*. URL: https://yukaichou.com/gamification-examples/octalysis-complete-gamification-framework/ (cit. on pp. 12, 14).

[19] Richard Bartle. «Hearts, clubs, diamonds, spades: Players who suit MUDs». In: *Journal of MUD research* 1.1 (1996), p. 19 (cit. on p. 14).

[20] René Eppmann, Magdalena Bekk, and Kristina Klein. «Gameful Experience in Gamification: Construction and Validation of a Gameful Experience Scale [GAMEX]». In: *Journal of Interactive Marketing* 43.1 (2018), pp. 98–115. DOI: 10.1016/j.intmar.2018.03.002. eprint: https://doi.org/10.1016/j.intmar.2018.03.002. URL: https://doi.org/10.1016/j.intmar.2018.03.002 (cit. on pp. 14, 15).

[21] Gordon Fraser. «Gamification of Software Testing». In: *2017 IEEE/ACM 12th International Workshop on Automation of Software Testing (AST)*. 2017, pp. 2–7. DOI: 10.1109/AST.2017.20 (cit. on p. 15).

[22] Gabriela Martins de Jesus, Fabiano Ferrari, Daniel Porto, and Sandra Fabbri. «Gamification in Software Testing: A Characterization Study». In: Sept. 2018, pp. 39–48. ISBN: 978-1-4503-6555-0. DOI: 10.1145/3266003.3266007 (cit. on p. 15).

[23] José Miguel Rojas and Gordon Fraser. «Code Defenders: A Mutation Testing Game». In: Apr. 2016, pp. 162–167. DOI: 10.1109/ICSTW.2016.43 (cit. on p. 15).

[24] Swapneel Sheth, Jonathan Bell, and Gail Kaiser. «HALO (Highly Addictive, Socially Optimized) Software Engineering». In: *Proceedings of the 1st International Workshop on Games and Software Engineering*. GAS '11. Waikiki, Honolulu, HI, USA: Association for Computing Machinery, 2011, pp. 29–32. ISBN: 9781450305785. DOI: 10.1145/1984674.1984685. URL: https://doi.org/10.1145/1984674.1984685 (cit. on p. 15).

[25] Natalia Silvis-Cividjian, Rob Limburg, Niels Althuisius, Emil Apostolov, Viktor Bonev, Robert Jansma, Glenn Visser, and Marc 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. Trondheim, Norway: Association for Computing Machinery, 2020, p. 553. ISBN: 9781450368742. DOI: 10.1145/3341525.3393975. URL: https://doi.org/10.1145/3341525.3393975 (cit. on p. 16).

[26] Peter J. Clarke, Andrew A. Allen, Tariq M. King, Edward L. Jones, and Prathiba Natesan. «Using a Web-Based Repository to Integrate Testing Tools into Programming Courses». In: *Proceedings of the ACM International Conference Companion on Object Oriented Programming Systems Languages and Applications Companion*. OOPSLA '10. Reno/Tahoe, Nevada, USA: Association for Computing Machinery, 2010, pp. 193–200. ISBN: 9781450302401. DOI: 10.1145/1869542.1869573. URL: https://doi.org/10.1145/1869542.1869573 (cit. on p. 16).

[27] Hoyama Santos, Vinicius Durelli, Maurício Souza, Eduardo Figueiredo, Lucas Silva, and Rafael Durelli. «CleanGame: Gamifying the Identification of Code Smells». In: Sept. 2019, pp. 437–446. ISBN: 978-1-4503-7651-8. DOI: 10.1145/3350768.3352490 (cit. on p. 16).

[28] Félix García, Oscar Pedreira, Mario Piattini, Ana Cerdeira-Pena, and Miguel Penabad. «A framework for gamification in software engineering». In: *Journal of Systems and Software* 132 (2017), pp. 21–40. ISSN: 0164-1212. DOI: https://doi.org/10.1016/j.jss.2017.06.021. URL: https://www.sciencedirect.com/science/article/pii/S0164121217301218 (cit. on p. 16).

[29] Filippo Cacciotto, Tommaso Fulcini, Riccardo Coppola, and Luca 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. DOI: 10.1109/ICSTW52544.2021.00032 (cit. on p. 17).

[30] Davide Gallotti. «Ideazione e prototipazione di un sistema di gamification per il testing di applicazione web». MA thesis. Politecnico di Torino, 2021. URL: https://webthesis.biblio.polito.it/19104/ (cit. on p. 17).

[31] Giacomo Garaccione, Tommaso Fulcini, and Marco Torchiano. «GERRY: A Gamified Browser Tool for GUI Testing». In: *Proceedings of the 1st International Workshop on Gamification of Software Development, Verification, and Validation*. Gamify 2022. Singapore, Singapore: Association for Computing Machinery, 2022, pp. 2–9. ISBN: 9781450394543. DOI: 10.1145/3548771.3561408. URL: https://doi.org/10.1145/3548771.3561408 (cit. on p. 17).

[32] Tommaso Sasso, Andrea Mocci, Michele Lanza, and Ebrisa Mastrodicasa. «How to gamify software engineering». In: Feb. 2017, pp. 261–271. DOI: 10.1109/SANER.2017.7884627 (cit. on p. 17).

[33] Ingrid Buckley and Peter Clarke. «An approach to Teaching Software Testing Supported by Two Different Online Content Delivery Methods.» In: Jan. 2018. DOI: 10.18687/LACCEI2018.1.1.377 (cit. on p. 18).

[34] Wishnu Prasetya et al. «Having Fun in Learning Formal Specifications». In: *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*. 2019, pp. 192–196. DOI: 10.1109/ICSE-SEET.2019.00028 (cit. on p. 18).

[35] *What is difference, limitation & use of Selenium, Cypress, or Appium framework?* URL: https://pharmasciences.in/cypress-vs-selenium-vs-appium/ (cit. on pp. 23, 24).

[36] Jash Unadkat. *Choosing the Right Automation Tool: Appium vs Selenium*. Feb. 2023. URL: https://www.browserstack.com/guide/appium-vs-selenium (cit. on p. 24).

[37] Jash Unadkat. *Cypress vs Selenium: Key Differences*. Feb. 2023. URL: https://www.browserstack.com/guide/cypress-vs-selenium (cit. on p. 24).

[38] Paolo Stefanut Bodnarescul. *Game GUI IntelliJ Plugin*. URL: https://github.com/Paolobd/intellij-gamification-plugin (cit. on p. 26).

[39] Paolo Stefanut Bodnarescul. *Game GUI Java Library*. URL: https://github.com/Paolobd/gamification-library (cit. on p. 26).

[40] *Comparison to Java*. URL: https://kotlinlang.org/docs/comparison-to-java.html (cit. on p. 27).

[41] *Di che cosa parliamo quando parliamo di gamification*. URL: https://learninglab.sdabocconi.it/files/sito/Gamification.pdf (cit. on p. 60).

[42] John Brooke. «SUS: A quick and dirty usability scale». In: *Usability Eval. Ind.* 189 (Nov. 1995) (cit. on p. 67).