POLITECNICO DI TORINO
Repository ISTITUZIONALE

GAppium

Original
GAppium / Laudadio, Lorenzo; Coppola, Riccardo; Torchiano, Marco; Fulcini, Tommaso. - ELETTRONICO. - (2025).

Availability:
This version is available at: 11583/3008534 since: 2026-03-10T14:50:03Z

Publisher:

Published
DOI:

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright

(Article begins on next page)

25 April 2026

GAppium

Lorenzo Laudadio, Riccardo Coppola, Marco Torhiano, Tommaso Fulcini
Dept. of Computer and Control Engineering
Politecnico Di Torino
Turin, Italy
first.last@email.com

March 1, 2025

Abstract

The testing process is a fundamental part of the software development cycle. One of the most used techniques to verify the proper functioning of mobile applications is Exploratory Testing. Exploratory Testing consists in a tester freely exploring the Graphical User Interface of the Application Under Test to detect anomalies and bugs. However, such a testing approach requires a large amount of time and has proven to be expensive and monotonous. A possible solution to mitigate the repetitiveness of human-performed activities is gamification. In recent years, several efforts have been made towards the introduction of gamification into the testing process. In this paper we provide a guide showing how to extend an existing open source Exploratory Testing tool by introducing gamification elements. The goal is to make the Exploratory Testing process more engaging and improve the testers' performances. We provide an example of a possible implementation of gamification dynamics: a progress bar that shows the tester the current progress. We believe that this kind of contribution can be useful and inspiring to the community working in the field.

Chapter 1

Introduction

Verification and validation is an essential part of the software development process. A widely used technique in the context of GUI-based verification is Exploratory Testing, consisting of the testers freely exploring the Application Under Test (AUT). This technique allows for simpler configuration of the testing environment and does not require writing code for test cases in parallel with production code development. At the same time, the testers have to be very experienced in order to choose actions that will trigger bugs and highlight software defects [1]. Moreover, the Exploratory Testing activity can easily become repetitive and, therefore, boring. Straubinger et al. [2] showed that one of the main causes of the limited amount of tests is the lack of testers' motivation. One technique often used to overcome the repetitiveness of tasks performed by human actors is gamification. Gamification, i.e. the application of game design mechanics to non-ludic contexts [3], includes a set of different tools – such as points, leaderboards, badges – that are used to elicit positive feelings in human subjects, prompting them to be more productive. In recent years, gamification has gained increasing interest in the field of software engineering [4, 5].

In this paper we introduce a framework aimed to develop gamification extensions integrated into the open-source testing tool Appium Inspector. In Section 2 we motivate our choices, giving also some contextual information and showing related work. Section 4 presents a sample extension we developed by leveraging our framework. We believe that this kind of example could be used as a guide for researchers and developers willing to produce similar extensions for mobile testing. Section 5 outlines the possible application and future research directions. In section 6 we summarize the contribution and explore the possible future work.

Acknowledgement:

This work was carried out within the "EndGame - Improving End-to-End Testing of Web and Mobile Apps through Gamification" project (CUP 2022PCCMLF) – funded by European Union – Next Generation EU Mission 4, Com-

ponent 1, within the PRIN 2022 program (D.D.104 - 02/02/2022 Ministero dell'Università e della Ricerca). This manuscript reflects only the authors' views and opinions and the Ministry cannot be considered responsible for them.

Chapter 2

Background and Related Work

The core idea of gamification is to improve the performances of the actors involved in some actions by making their tasks more appealing and enjoyable. One of the most used frameworks for the application of Gamification is the Octalysis Framework [6]. The Octalysis Framework defines 8 Core Drives (CDs) commonly found in games, that can be used to increase the motivation of the user. In particular, for the development of our sample extension, we focused on two core drives:

- **CD2**: Development & Accomplishment. This CD is associated with the sense of personal growth that is experienced when makingp progress, developing skills, and eventually overcoming challenges.
- **CD4**: Ownership & Possession. This CD is related to the motivation users have when they feel they own something, and is typically implemented through customization.

We used some of the metrics defined by Cacciotto et al. [7], like the page coverage – computed as the percentage of widgets interacted on the current page – and a modified version of the session coverage: the incremental page coverage – computed as the percentage of widgets interacted over all the pages discovered in the current session.

Gamification applied to software testing has given promising results in the past. Fraser obtained positive results by applying gamification in a crowdsourcing environment [8]. Straubinger et al. integrated gamification elements into the Jenkins continuous integration tool [9]. Fulcini et al. highlighted the benefits of gamification applied to Exploratory GUI testing with a preliminary evaluation [10]. In general, game design mechanics seem particularly suitable to increase the engagement of testers while conducting their activities [11].

Chapter 3

Instruments

3.1 Electron

Electron is a JavaScript framework used to build standalone desktop applications [12]. The Electron execution model encompasses two processes: the Main and Renderer process. The Main process has access to a full Node.js environment, while the Renderer process is responsible for rendering web content.

3.2 React

React is a very popular Web-UI framework to build user interfaces with the JSX markup language, a mixture of HTML and JavaScript [13]. Everything in React is based upon the concept of Components, reusable functions that take properties as input. Components are (re-)rendered whenever something in their properties changes. In addition, React defines Hooks: functions that can be used to add functionalities to the basic behavior of components.

3.3 Redux

The React framework is itself capable of managing a state that is local to the components. While this approach can be suitable for small projects, when the size of the application grows, it becomes unmanageable. In the worst case, it is possible to have a distinct state for each component.

Redux is a popular JavaScript framework that solves this problem. It defines a global state (or store) and imposes the so-called "one-way data flow", depicted in Figure 3.1 [14]. The Redux state is read-only, and the only way to change it is through the dispatch method of the state which takes *actions* as input, i.e. objects which describe the event that happened, and through *reducers*, pure functions which take as input the previous state and an action and return a new

Figure 3.1: Redux - One Way Data Flow

state. This functional and neat approach leads to cleaner code and predictable results.

Chapter 4

GAppium

The proposed gamified extension for Appium Inspector is an almost trivial tool. In fact, it servers the main purpose of illustrating the steps required to implement a gamified extensions on top of Appium Inspector.

4.1 Basic Concepts and Gamification Info

Figure 4.1 shows the main folders within the Appium Inspector directory tree. The `actions/` folder contains the code for the actions, the `reducers/` folder contains the code for the reducers, etc.

Appium Inspector defines two application contexts: the *Session* and the *Inspector*. The Session context is the one in which you define the Session properties, like the remote address of the Appium Server, the remote port, the desired capabilities, etc. The Inspector, instead, is a particular running instance of a Session, and provides all the tools needed to interact with the running mobile application.

Our idea was to implement two simple gamification mechanics:

- **Progress bars** showing the current page coverage and the incremental page coverage (related to CD2 from the Octalysis framework).
- A simple username TextInput with an avatar icon representing the **user** (related to CD4 from the Octalysis framework).

We do not need to modify the Session files: we just need to modify the behavior of the Inspector. The Inspector page already contains different components (tabs) that allow performing different actions on the current AUT (Source, Commands, Gestures, Recorder, Session Information). Our goal is to reuse as much as possible of the existing GUI, according to the React principles. In this case, we can adopt a bottom-up approach, starting from the GUI and then adding the needed functionalities.

We define a new tab for the inspector, called *GamificationInfo*, which will allow the user to interact with the application and get information about the

Figure 4.1: Appium Inspector Folder Tree

current session. The creation of a new tab in Appium Inspector can be made by creating a *.jsx* file named after the tab under the `components/Inspector` folder. The SessionInfo component already features a table that displays info about the server details, session details, etc. From this tab, we can take the session details about the currently active app ID and the session length. The table entries are defined within a *.js* file under the `/constants` folder.

4.2 Adding the progress bars

The next step is to display the page coverage progress bars. To compute the page coverage we first need to define what a page is, since mobile applications do not have this notion internally. In this case, we built a simple algorithm to compute the current page ID: we concatenate the `class` attributes of the widgets present on the screen. This allows for easy comparison between pages. This algorithm for page identification can be substituted – in future work – with more refined ones that would allow to consider cases in which the page layout is modified at runtime.

The page ID needs to be rebuilt whenever the source code of the page changes. The source code of the page is already stored in the sourceJSON field of the Redux state, and each component has access to it through the `props` variable. React exposes the `useEffect` Hook, which allows us to define a callback that gets called whenever the sourceJSON field changes. Our callback should:

- Build the current page ID
- Add a page object to the state's pages array, if a page with the same ID is not already present
- Set the current page ID in the state

The code of the `useEffect` is shown in Figure 4.2.

The functions `addPage` and `setCurrentPageId` modify the state. This means that we need to provide support via actions and reducers.

First of all, we need to define two new types of actions. The types of actions are defined in the `renderer/actions/Inspector.js` file. As described in Section 2, actions are objects that describe the happening of an event, where events are typically triggered by an interaction with GUI. The only way to modify the state is by calling the dispatch method of the state with an action object.

```
// components/Inspector/GamificationInfo.jsx
useEffect(() => {
  const pageId = buildPageId(sourceJSON);
  if(! pages.some(p => p.pageId == pageId)){
    const newPage = {
      pageId: pageId,
      nInteractableWidgets: countWidgets(sourceJSON),
      nInteractedWidgets: 0 };
    addPage(newPage);
  }
  setCurrentPageId(pageId);
}, [sourceJSON]);
```

Figure 4.2: The useEffect Hook

```
// renderer/actions/Inspector.js
export function setCurrentPageId(pageId){
  return (dispatch) => {
    dispatch({type: SET_PAGE_ID, pageId});
  }
}

export function addPage(page){
  return (dispatch) => {
      dispatch({type: ADD_PAGE, page});
  }
}
```

Figure 4.3: Actions

Whenever an action gets dispatched, the reducer updates the part of the state associated with the action.

To summarize, here is the chain of events which leads to the new state:

1. The sourceJSON gets updated.
2. The useEffect hook runs, by calling the addPage and setCurrentPageId functions.
3. The two functions call the dispatch method of the state with the appropriate actions.
4. The reducer function receives the dispatched actions and returns a new version of the state, with the corresponding fields updated.

A simplified version of the code for the actions and the reducers is shown in Figure 4.3 and Figure 4.4.

A similar approach can be used to increment the number of interacted widgets within the current page: we define a new type of action create a function to dispatch the action, update the renderer with the code needed to update the fields of the current page. The current page ID in this case is useful to retrieve the current page from the array of pages in the state that we just added.

Now that we have added the needed functionalities, it is sufficient to update the GUI with the missing components. For the progress bars, we used the Progress component from the Ant Design library, the GUI library used by Appium Inspector, which receives as input a number and draws a progress bar of proportional length. The code is shown in Figure 4.5. Figure 4.6 shows an implementation of Gamification tab with two progress bars, one for the current

```
// renderer/reducers/Inspector.js
export default function inspector(state=initial_state, action){
  switch (action.type) {
    // ...
    case ADD_PAGE:
      return {
        ...state,
        pages: [...state.pages, action.page],
      }

    case SET_PAGE_ID:
      return {
        ...state,
        currentPageId: action.pageId
      }
}
```

Figure 4.4: Reducers

```
// components/Inspector/GamificationInfo.jsx
const getCurrentPageCoverage = () => {
  if(currentPageId){
    const page = pages.find(p => p.pageId === currentPageId);
    return Math.round(100* page.interactedWidgets/ page.interactableWidgets)
  } else
    return 0;
}
// ...
</Progress percent={currentPageId ? 0 : getCurrentPageCoverage()}>
// ...
```

Figure 4.5: Progress Bar Component

page coverage and the other for the incremental page coverage, the source code of the page and a TextInput in which testers can insert their names near an avatar icon[1].

[1]The complete source code will be made available through Zenodo.

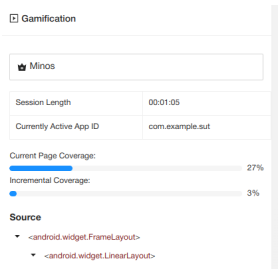

Figure 4.6: Screenshot of GAppium

Chapter 5

Application and Research Directions

We believe that our work could serve as a reference guide for the development of many other gamified extensions. The advantage of working on an already existing open-source, cross-platform application is indisputable. The approach described in this paper, and the source code in the replication, is considered an aid for test tool developers and researcher to enhance the capabilities of Appium Inspector without the need of exploring the entire codebase.

Future research directions include the possibility of integrating our extension with a leaderboard system that allows testers to compete with each other or against themselves. This could be mixed with a point assignment system. For example, testers could receive a certain amount of points for the number of bugs found.

It would be possible to limit the session time to a fixed duration, like, for example, 10 minutes. At the end of each session, the progress of the testers could be saved and logged within the leaderboards.

Additionally, the page detection algorithm can be improved substantially with respect to the one presented here. For example, a metric of similarity between pages could be defined, so that if the layout of a page changes slightly the application is still able to recognize it as the same page [15].

These enhancements can easily be integrated with existing features of Appium Inspector, like for instance the test capture & replay feature, which allows recording JUnit test cases.

Chapter 6

Conclusion and Future Work

In this paper, we presented Gappium, a framework for the development of gamified extensions to the mobile testing tool Appium Inspector. In a sample extension we implemented two different gamification mechanics: progress bars showing the session coverage and a customizable username for the tester, with an avatar icon. These mechanics are commonly used to enhance the testers' motivation through their testing sessions and to make their tasks more enjoyable. Further work is in progress involving the integration of different gamification mechanics into the tool, and the evaluation of the application both from the point of view of usability – e.g. with tools like System Usability Scale (SUS) [16], GAMEX [17] or Technology Acceptance Models (TAM) [18] – and the effectiveness perspective. In conclusion, we believe that our work could be foundational for future research, simplifying considerably the development process of similar gamified extensions.

Bibliography

[1] S. Ozturk, "Gamification of exploratory testing process," in *Proceedings of the 1st International Workshop on Gamification of Software Development, Verification, and Validation*, Gamify 2022, (New York, NY, USA), pp. 14–17, Association for Computing Machinery, Nov. 2022.

[2] P. Straubinger and G. Fraser, "A Survey on What Developers Think About Testing," in *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*, pp. 80–90, Oct. 2023. ISSN: 2332-6549.

[3] S. Deterding, R. Khaled, L. E. Nacke, D. Dixon, *et al.*, "Gamification: Toward a definition," in *CHI 2011 gamification workshop proceedings*, vol. 12, pp. 1–79, Vancouver, 2011.

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

[5] C. F. Barreto and C. França, "Gamification in Software Engineering: A literature Review," in *2021 IEEE/ACM 13th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*, pp. 105–108, May 2021.

[6] Y. Chou, *Actionable Gamification: Beyond Points, Badges, and Leaderboards*. Octalysis Media, 2019.

[7] F. Cacciotto, T. Fulcini, R. Coppola, and L. Ardito, "A Metric Framework for the Gamification of Web and Mobile GUI Testing," in *2021 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pp. 126–129, Apr. 2021.

[8] G. Fraser, "Gamification of Software Testing," in *2017 IEEE/ACM 12th International Workshop on Automation of Software Testing (AST)*, pp. 2–7, May 2017.

[9] P. Straubinger and G. Fraser, "Gamekins: gamifying software testing in jenkins," in *Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Companion Proceedings*, ICSE '22, (New York, NY, USA), pp. 85–89, Association for Computing Machinery, Oct. 2022.

[10] T. Fulcini and L. Ardito, "Gamified Exploratory GUI Testing of Web Applications: a Preliminary Evaluation," in *2022 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pp. 215–222, Apr. 2022. ISSN: 2159-4848.

[11] T. Fulcini, R. Coppola, L. Ardito, and M. Torchiano, "A Review on Tools, Mechanics, Benefits, and Challenges of Gamified Software Testing," *ACM Computing Surveys*, vol. 55, pp. 310:1–310:37, July 2023.

[12] "Electron - build Cross-Platform Desktop apps with JavaScript, HTML, and CSS," 2024.

[13] "React - the Library for Web and native User Interfaces," 2024.

[14] "Redux - a JS Library for predictable and maintainable global State management," 2024.

[15] T. Fulcini, R. Coppola, M. Torchiano, and L. Ardito, "An analysis of widget layout attributes to support android gui-based testing," in *2023 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pp. 117–125, 2023.

[16] R. A. Grier, A. Bangor, P. Kortum, and S. C. Peres, "The system usability scale: Beyond standard usability testing," *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, vol. 57, no. 1, pp. 187–191, 2013.

[17] R. Eppmann, M. Bekk, and K. Klein, "Gameful experience in gamification: Construction and validation of a gameful experience scale [gamex]," *Journal of Interactive Marketing*, vol. 43, no. 1, pp. 98–115, 2018.

[18] F. D. Davis and A. Granić, *The Technology Acceptance Model: 30 Years of TAM*. Human–Computer Interaction Series, Cham: Springer International Publishing, 2024.