Mobile GUI test script generation from natural language descriptions using pre-trained model

Chun Li
chunli@smail.nju.edu.cn
State Key Laboratory for Novel Software Technology, Nanjing University
Software Institute, Nanjing University
Nanjing, Jiangsu, China

ABSTRACT

GUI test scripts are valuable assets to guarantee the quality of mobile apps; however, manually writing executable GUI test scripts can incur huge cost. In this paper, we propose an approach to the generation of test scripts from the natural language descriptions, with the help of descriptions to locate elements and use the attributes of elements to select actions to construct the corresponding events. The construction of test scripts with the help of natural language descriptions can greatly reduce the burden of testers and is robust to changes in the position of GUI elements.

ACM Reference Format:
Chun Li. 2022. Mobile GUI test script generation from natural language descriptions using pre-trained model. In *IEEE/ACM 9th International Conference on Mobile Software Engineering and Systems (MOBILESoft '22), May 17–24, 2022, Pittsburgh, PA, USA.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3524613.3527809

1 INTRODUCTION

In the last decade, with the popularity of mobile devices, the demand for mobile apps has been increasing. Among the many mobile platforms, the highest share is on Android. There are already more than two and a half million Android applications in the Google Play. It is imperative to guarantee the quality of mobile apps.

Compared with traditional software testing, GUI testing has been the dominant testing approach for mobile apps. The automation of testing is attractive, and there are already many tools. However, it is difficult for these tools to consider the tester's intention regarding the app logic, and thus, although they substantially reduce the tester's burden, the testing efficiency is lower than that of manually written GUI test scripts.

While GUI test scripts can be automatically executed using tools such as Appium or Robotium, the central task of writing GUI test scripts—to locate the GUI elements and then perform operations on them such as click and swipe—still requires manual effort. It can be a tedious and time-consuming task. Even worse, GUI test scripts become unavailable during version iterations, as elements may be added, deleted, or changed in position on the GUI. Therefore, GUI test scripts are fragile.

In the process of writing test scripts, testers will often code their scripts based on some documents. For example, when a use case in the document requires the creation of a contact, the tester will look for an element in the corresponding GUI context that can accomplish the function and then perform an action on the element. These descriptions of events contain essential content related to the application logic and GUI elements.

This inspires us to generate the test events in a script from the natural language descriptions of the events. We propose a method that uses SBERT [6] to compute the semantic similarity between the natural language description of the event and the attributes of GUI elements to determine the specific element corresponding to the event to be executed. Then we derive the test action of the event from the type of the located element. In this way, the tester only needs to write the natural language descriptions for the test script, which not only improves readability but also significantly reduces the cost of manually writing and maintaining test scripts.

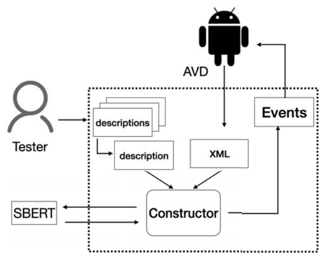
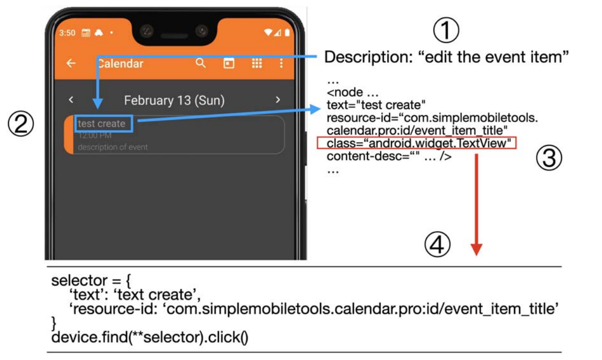


Figure 1: Architecture of the approach

2 APPROACH

Before presenting our approach, we first define *test event* (*event* for short). An *event* is an action executed on an element of the GUI under test, i.e., $Event := \{Selector, Action\}$. The *Selector* is a dictionary containing the **text**, **content-desc** and **resource-id** properties of the GUI element, which is used to locate the element on the current GUI, and *Action* is the test action that can be performed on the element, such as **Click**, **Swipe**, etc.

Figure 1 presents the Architecture of our approach. In Figure 1, *Description* is the natural language description of the event that the tester intends to trigger. It contains the content related to the app logic. In addition, each *description* can only correspond to one but not multiple events. To locate the elements, we first calculate the semantic similarity between the description and all the elements on the current GUI. The semantic similarity between



```
...
<node ...
text="test create"
resource-id="com.simplemobiletools.
calendar.pro:id/event_item_title"
class="android.widget.TextView"
content-desc="" ... />
...
```

```
selector = {
    'text': 'text create',
    'resource-id: 'com.simplemobiletools.calendar.pro:id/event_item_title'
}
device.find(**selector).click()
```

Figure 2: build script from description

the description and the element is sorted by *confidence*. We input the text, content-desc, and resource-id [1] of the element into SBERT [6] to calculate the semantic similarity with the description and select the maximum of them as the confidence of the element and the rest of elements will be used as candidates. Particularly, if resource-id is a combination of package and id, only id is selected; and if there is an empty attribute, it is not input for calculation.

After locating the element, we devise the test action according to the type of the elements. Since most of the actions in the description are related to the app logic, their descriptive words and the actions that the mobile app accepts can be very different, such as **create** and **click**. There can be almost no semantic similarity, making it difficult to translate directly from descriptive words to actions. Still, we can construct events corresponding to the type of the element, since there is a limit to the actions that a specific type of element can accept. For example, if the type of the element is a button, we can try to combine it with click/double click, etc.

Figure 2 shows how we generate the script using our approach. First, we take the description "**edit the event item**" and the XML of the current GUI to locate the element. The element with the highest confidence has been marked with the **blue** box, and on the right is its properties (only key information is shown). According to the **Class** information in the **red** box, we know that its type is **TextView** and choose the **click** action. Combining the action with the attributes, we get a test script shown in the bottom of the figure.

If a test action cannot be constructed on the current GUI or the package of the current GUI is different from the app under test, our tool will perform *tracing*. It will uninstall and reinstall the app and re-execute the events except the event causing tracing. Assuming that the events have been executed are $Event_1, \ldots, Event_i$, our method will re-execute the $Event_1, \ldots, Event_{i-1}$, for $Event_i$, assuming it is constructed by d_i descriptions , and the events constructed by d_i is S_i, we will select the event with the highest confidence from S_i to replace $Event_i$ (note that $Event_i$ is not in S_i), then execute it. If executing the new event still results in tracing, repeat the above process. When all events in S_i have been tried, it will trace to $Event_{i-1}$, and then continue the above process.

3 CASE STUDY

We conduct a preliminary case study on three open-source apps and prepare test descriptions ourselves. The result of our case study is shown in Table 1. We record the following information:

project	NoS	NoD	AoD	SCD	SES
Calendar	3	16	5.3	13	2
Wallet	2	15	7.5	13	1
AcDisplay	2	12	6	11	1

Table 1: Result

- **NoS**: number of scripts
- **NoD**: number of descriptions
- **AoD**: average number of descriptions per script
- **SCD**: successfully constructed description
- **SES**: successfully executed script

As Table 1 shows, our approach successfully constructed 37 test actions from 43 descriptions, resulting in an 86% success rate. We also studied the reasons for the incorrect constructed actions. As Table 1 shows, our approach successfully constructed 37 test actions from 43 descriptions, resulting in an 86% success rate. We also studied the reasons for the incorrect constructed actions. In Calendar, an incorrect construction is caused by selecting a wrong element, resulting in an error in the construction of the subsequent action. The reasons for Wallet is more interesting. For example, one incorrect action takes the parent element of the correct element because the parent element has the needed properties while the correct element is missing these properties. We will improve our approach based on the study of these incorrections in the future.

4 RELATED WORK

The recording technique is a way to generate test scripts [3–5], with the disadvantage that some events cannot be recorded and lack robustness to evolution of apps. Work [2] uses machine learning and formal description to generate robust and reusable test scripts. Our work differs from his in that we only require descriptions that are relevant to the app logic and do not require formalization.

5 CONCLUSION

We propose a method for generating test scripts based on natural language descriptions using pre-trained models. While this method has high accuracy in constructing events, it does not require too many restrictions on description, which can reduce the burden of testers.

REFERENCES

[1] Google. 2022. guide of resources. https://developer.android.com/guide/topics/resources/providing-resources.
[2] Gang Hu, Linjie Zhu, and Junfeng Yang. 2018. AppFlow: using machine learning to synthesize robust, reusable UI tests. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 269–282.
[3] Yongjian Hu and Iulian Neamtiu. 2016. VALERA: an effective and efficient record-and-replay tool for android. In *Proceedings of the International Conference on Mobile Software Engineering and Systems*. 285–286.
[4] Wing Lam, Zhengkai Wu, Dengfeng Li, Wenyu Wang, Haibing Zheng, Hui Luo, Peng Yan, Yuetang Deng, and Tao Xie. 2017. Record and replay for android: Are we there yet in industrial cases?. In *Proceedings of the 2017 11th joint meeting on foundations of software engineering*. 854–859.
[5] Stas Negara, Naeem Esfahani, and Raymond Buse. 2019. Practical android test recording with espresso test recorder. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 193–202.
[6] Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084* (2019).