Measuring the Energy Footprint of Mobile Testing Frameworks

Luis Cruz
University of Porto / INESC ID
Lisbon, Portugal
luiscruz@fe.up.pt

Rui Abreu
University of Lisbon / INESC ID
Lisbon, Portugal
rui@computer.org

ABSTRACT

This paper evaluates eight popular mobile UI automation frameworks. We have discovered that there are automation frameworks that increase energy consumption up to 7500%. While limited in the interactions one can do, *Espresso* is the most energy efficient framework. Depending on the needs of the tester, *Appium*, *Monkeyrunner*, or *UIAutomator* are good alternatives. We show the importance of using energy efficient frameworks and provide a decision tree to help developers make an educated decision on which framework suits best their testing needs.

CCS CONCEPTS

• **General and reference → Empirical studies**; • **Software and its engineering → Software performance**;

KEYWORDS

Mobile Testing; Testing Frameworks; Energy Consumption.

ACM Reference Format:
Luis Cruz and Rui Abreu. 2018. Measuring the Energy Footprint of Mobile Testing Frameworks. In *Proceedings of 40th International Conference on Software Engineering Companion, Gothenburg, Sweden, May 27-June 3, 2018 (ICSE '18 Companion),* 2 pages.
https://doi.org/10.1145/3183440.3195027

1 INTRODUCTION

The popularity of mobile applications (also known as *apps*) have brought a unique, non-functional concern — energy efficiency [2]. It is important to provide developers with tools and knowledge to ship energy efficient mobile apps [3–5].

UI automation frameworks are used in the context of energy efficiency to mimic user interaction in an application while using an energy profiling tool [1]. However, there are still energy-related concerns that need to be addressed [6].

As a motivational example, consider the following scenario: a developer needs to optimize the energy efficiency of an app that provides a list of *tweets* featuring media content. The developer decided to show plain text and pictures with low resolution. Original media content would be rendered upon a user *Tap*. This solution was validated by creating automated interaction scripts to mimic user interaction and profile energy consumption. In the end, the script for the new version had five extra *Taps*. Hence, the automation framework will be spending more energy to perform these extra interactions. Imagining that for each *Tap* the automation framework spends 1 millijoule (mJ), the new version will have to spend at least $5mJ$ less than the original version in order to be perceived as being more efficient. More efficient frameworks could reduce this threshold to a more insignificant value. However, since automation frameworks have not considered energy consumption as an issue, developers do not have a sense of which framework is more suitable for the task.

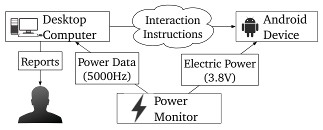

Figure 1: Experimentation system.

The main goal of this work is to study popular UI automation frameworks in the context of testing the energy efficiency of mobile apps. We measure the energy consumption of common user interactions: *Tap*, *Long Tap*, *Drag And Drop*, *Swipe*, *Pinch & Spread*, *Back button*, *Input text*. In addition, helper methods are also studied: *Find by id*, *Find by description*, and *Find by content*. The state-of-the-art UI automation frameworks for Android used in our study are *Appium*, *UIAutomator*, *PythonUIAutomator*, *AndroidViewClient*, *Espresso*, *Robotium*, *Monkeyrunner*, and *Calabash*.

Results show that *Espresso* is the framework with best energy footprint, although *Appium*, *Monkeyrunner*, and *UIAutomator* are also good candidates. On the other side of the spectrum are *AndroidViewClient* and *Calabash*, which make them not suitable to test the energy efficiency of apps yet. We have further discovered that methods that use content to look up UI components need to be avoided since they are not energy savvy.

2 DESIGN OF THE EMPIRICAL STUDY

We leverage the experimental setup illustrated in Figure 1, with three main components: a desktop computer that serves as controller; a power monitor, i.e., the Monsoon's *Original Power Monitor*, to measure energy consumption; and an Android device.

A script for every pair of framework and interaction was created, yielding 73 scripts. Each script calls the the method in a framework that mimics a given user interaction. Each experiment was identically and independently repeated 30 times. We use a self developed Android app to prevent any action that could possibly be triggered by the events of user interaction. With this app we assure that the energy consumption is not affected by the operations of an ordinary app.

3 RESULTS

Table 1 presents the percentage overhead of a framework when compared to the baseline. A long dash (—) marks the most efficient framework for a given interaction, being used as baseline. Frameworks that do not support a given interaction are marked with *n.a.*. Results can be visualized with the plot of Figure 2 — each bar presents the mean energy consumption of executing one UI interaction.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
ICSE '18 Companion, May 27-June 3, 2018, Gothenburg, Sweden
© 2018 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-5663-3/18/05.
https://doi.org/10.1145/3183440.3195027

Table 1: Overview of the studied UI automation frameworks

	Tap	Find By Id	Find By Content	Swipe	Back Button	Find By Description	Dragndrop	Input Text	Pinch And Spread	Long Tap
AndroidViewClient	224.6%	2716.9%	1163.9%	337.6%	4548.9%	2691.3%	99.3%	7474.6%	n.a.	154.7%
Appium	801.4%	332.2%	—	65.2%	31.8%	369.2%	201.9%	1415.4%	n.a.	27.5%
Calabash	381.7%	2897.4%	935.3%	318.1%	1578.0%	2933.0%	826.0%	825.6%	349.7%	136.3%
Espresso	—	—	207.2%	—	382.2%	—	n.a.	133.0%	n.a.	—
Monkeyrunner	197.7%	99.5%	n.a.	254.0%	—	n.a.	96.8%	110.9%	n.a.	9.1%
PythonUiAutomator	50.7%	512.3%	0.9%	155.3%	2806.8%	384.5%	150.7%	212.5%	187.3%	142.5%
Robotium	140.2%	1935.0%	673.6%	299.1%	3176.7%	n.a.	42.5%	58.1%	n.a.	98.0%
UiAutomator	25.8%	282.6%	14.1%	166.3%	2603.6%	275.9%	—	—	—	12.7%

4 CONCLUSION

Specific UI interactions have distinct energy consumptions depending on the framework. Increases on energy consumption are observed to go up to 7500%, as is the case of *Input Text*.

One thing that stands out from Figure 2 is the fact that looking up one UI component is expensive. The result of calling lookup methods should be cached whenever possible. In addition, lookup methods based on content need to be avoided since they consistently yield poor energy efficiency.

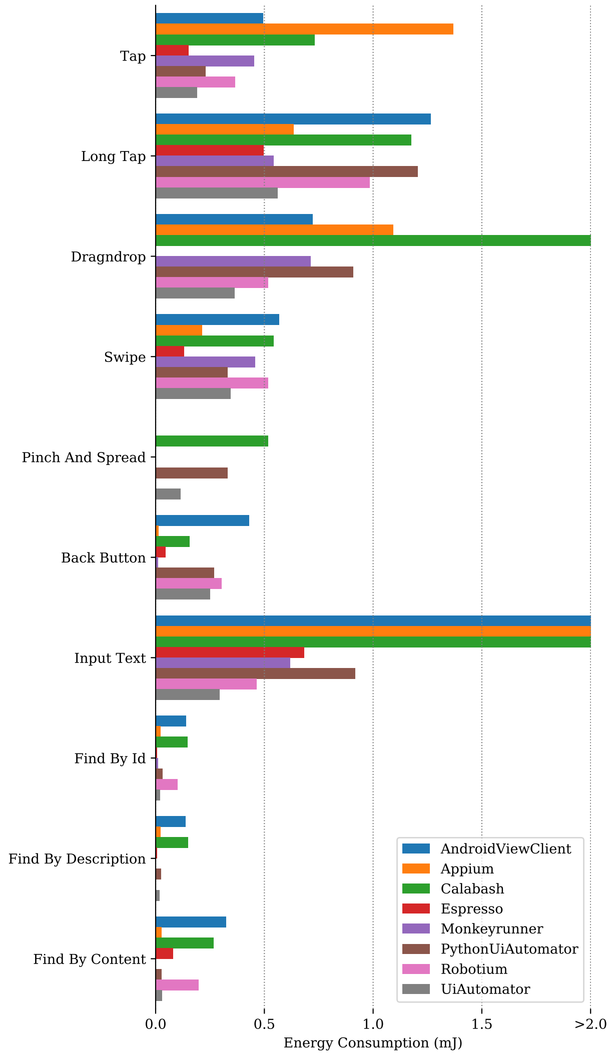

Figure 2: Energy consumption of UI interactions.

Results show that Espresso consistently yielded good results, and that *AndroidViewClient* and *Calabash* are not suitable for energy consumption profiling. However, choosing the right framework for a project can be challenging: there is no *one solution fits all*. Based on our observations, Figure 3 depicts a decision tree to help software developers decide which framework is most suited for their project.

This work paves the way for the inclusion of energy tests in the development stack of apps. It brings awareness on the energy footprint of tools used for energy test instrumentation, affecting both academic and industrial use cases.

5 ACKNOWLEDGMENTS

This work is financed by the ERDF — European Regional Development Fund through the Operational Program for Competitiveness and Internationalization - COMPETE 2020 Program and by National Funds through the Portuguese funding agency, FCT - Fundação para a Ciência e a Tecnologia within project POCI-01-0145-FEDER-016718. Luis Cruz is sponsored by an FCT scholarship grant number PD/BD/52237/2013.

REFERENCES

[1] Luis Cruz and Rui Abreu. 2017. Performance-based Guidelines for Energy Efficient Mobile Applications. In *IEEE/ACM International Conference on Mobile Software Engineering and Systems, MobileSoft 2017*. 46–57.

[2] Denzil Ferreira, Eija Ferreira, Jorge Goncalves, Vassilis Kostakos, and Anind K Dey. 2013. Revisiting human-battery interaction with an interactive battery interface. In *Proceedings of the 2013 ACM international joint conference on Pervasive and ubiquitous computing*. ACM, 563–572.

[3] Mario Linares-Vásquez, Kevin Moran, and Denys Poshyvanyk. 2017. Continuous, evolutionary and large-scale: A new perspective for automated mobile app testing. In *Software Maintenance and Evolution (ICSME), 2017 IEEE International Conference on*. IEEE, 399–410.

[4] Candy Pang, Abram Hindle, Bram Adams, and Ahmed E Hassan. 2016. What do programmers know about software energy consumption? *IEEE Software* 33, 3 (2016), 83–89.

[5] Rui Pereira, Tiago Carção, Marco Couto, Jácome Cunha, João Paulo Fernandes, and João Saraiva. 2017. Helping programmers improve the energy efficiency of source code. In *Proceedings of the 39th International Conference on Software Engineering Companion*. IEEE Press, 238–240.

[6] Kent Rasmussen, Alex Wilson, and Abram Hindle. 2014. Green mining: energy consumption of advertisement blocking methods. In *Proceedings of the 3rd International Workshop on Green and Sustainable Software*. ACM, 38–45.

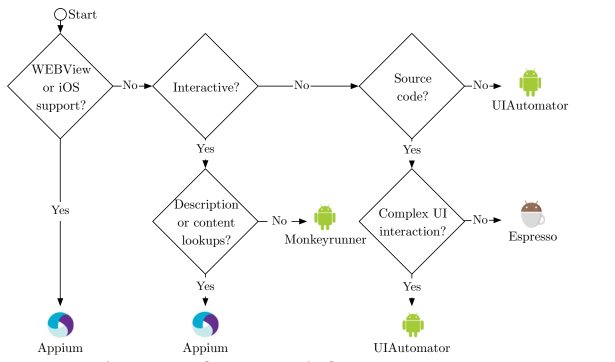

Figure 3: Selecting a framework for energy measurements.