SAUCELABS

E-BOOK

Mobile App Testing: Main Challenges, Different Approaches, One Solution

TABLE OF CONTENTS

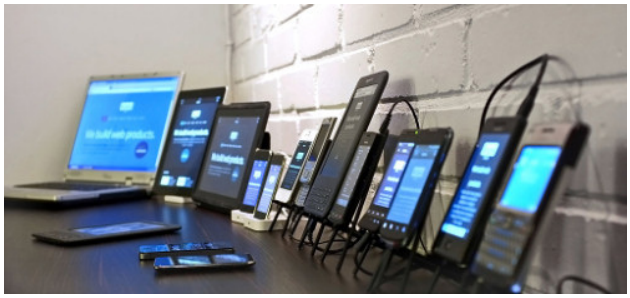
1. INTRODUCTION

In a heavily fragmented mobile environment, developing applications, which are compatible with multiple platforms and meet increased user expectations, represents a big challenge. Good user experience is expressed through ratings and rankings in the app stores, which directly influences the lifecycle of any app and determines its failure or success rate. Therefore, assuring an app's compatibility and functionality is crucial, as is using different test scenarios.

Many organizations are still struggling to implement effective testing approaches that fit with current software development processes. Most of today's available testing methodologies are based on traditional waterfall development methods. While more than 30% of organizations have already implemented the agile development methodology to some degree, just 9% of organizations are fully agile [26].

Although agile projects show an overall higher percentage of success rates, an essential success factor for agile projects is to have the right skill set within the application delivery team. Lacking the right test automation tools means that teams of developers and testers spend excessive time in testing manually and having to cover multiple devices, operating systems, and various configurations.

Cloud infrastructure presents an optimal solution for businesses who are looking for convenient and accessible tools that fit into their development processes without setting up their own infrastructure. Enabling testing on hundreds of real devices, cloud-based automated testing services can provide an almost instant reduction of downtime, which lowers costs, and helps deliver applications to the market faster.



This paper will discuss different challenges in mobile application testing and will introduce advantages and disadvantages of manual testing as well as test automation. In addition, this paper will give an overview of different open source frameworks and suitable test infrastructures. Finally, this report will suggest ways to optimize mobile application testing and show how Sauce Labs can overcome challenges and inefficiencies.

2. WHAT IS MOBILE APPLICATION TESTING AND WHY IS IT IMPORTANT?

The number and variety of consumer and enterprise mobile applications has grown exponentially over the last few years. In December 2016, Android users were able to choose between 2.6 million apps, while Apple's App Store remained the second-largest app store with 2 million available apps [1].

But even though release cycles become shorter, mobile devices and platforms are extensively moving towards fragmentation and usability requirements have become more complex. Mobile users are not forgiving and no company can afford a bad rating in the app stores. To prevent revenue loss, lost productivity and damage to brand reputation, organizations need to ensure that every application runs flawlessly and meets the highest quality standards. This is where mobile app testing comes in.

In general, **mobile application testing** can be described as the process by which application software developed for handheld mobile devices is tested for functionality, usability and consistency [2].

Apart from functional testing, which – most importantly – ensures that the application is working per the requirements, performance testing (behavior & performance under certain conditions; bad connections or low battery), security or compliance testing are also very important [3]. Usability is one of the keys to commercial success and therefore testing is crucial to verify if the application is achieving its goals and if it is getting a favorable response from users.

For the best possible solution to get applications to market on time and within budget, a comprehensive testing strategy is required. This mobile testing strategy should include a device and a network infrastructure, an optimized selection of target devices, and an effective combination of manual and automated testing tools to cover both functional and non-functional testing [4].

3. WHAT ARE THE MAIN CHALLENGES OF MOBILE APP TESTING?

As already stated, mobile application testing is different and more complex than testing traditional desktop and web applications and has its own set of (new) challenges [5]. We will now review the main challenges.

First of all, the **increasing fragmentation of mobile devices** marks a major problem for mobile application developers and testers. The Android Fragmentation Report 2015 [6] states that there had been 24,093 distinct Android devices until August 2015 (compared to 11,868 devices in 2013). All of those Android devices differ in their shapes and sizes, with vastly different performance levels, screen sizes and input methods with different hardware capabilities.

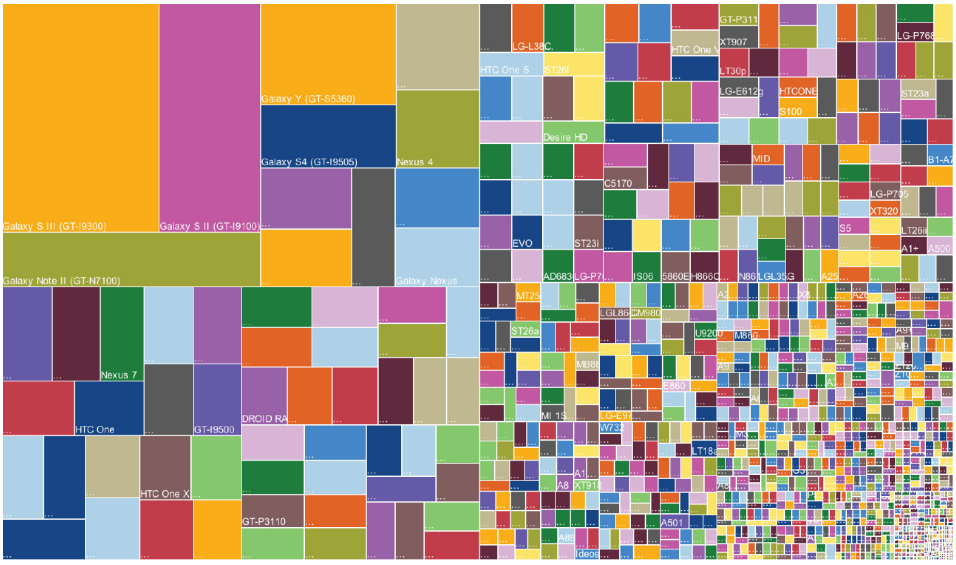


Figure 1: Device Fragmentation: 24,093 distinct Android devices had been seen until August 2015 [6].

Also the **overwhelming diversity in mobile platforms and operating systems (OS)** poses a unique challenge for testers. Not only are there many different OSs with their own limitations available in the market, but also a great variety of (older or recent) versions of the same OS. This phenomenon is perfectly described within Margaret Rouse's device fragmentation definition on TechTarget. For her, mobile device fragmentation is "[...] a phenomenon that occurs when some mobile users are running older versions of an operating system, while other users are running newer versions" [7].

In addition, the **fast release cycles** of mobile applications make it difficult for QA teams to ensure high quality standards for the apps. This can only be made possible through test automation and regression testing (the process of testing changes to computer programs to make sure that the older programming still works with the new changes [8]).

Another challenge in testing mobile application is represented by the **huge number of Mobile Network Operators**. At the moment there are more than 670 Mobile Network Operators in the world, using different network standards and different kinds of network infrastructure.

Furthermore, the **mobility of users and the fact that they are moving around** while using apps can also be seen as a problem for app developers and testers, since they require an internet connection to fetch data and serve the user with updates and information. Daniel Knott illustrates this challenge by a striking example of an application that tracks the speed of snowboarders. Even if there is a good network connection with high speed at the top of the mountain, he writes, this does not mean that the connection will be good down in the valley. Whether the app crashes or still works must also be considered in test scenarios [9].

Finally, the **international use of applications** marks an important challenge in mobile testing. As many apps are developed for international markets – apart from the mere translation of contents – regional traits, time zones and target audiences must be taken into account. Also, Arabic script or other right-to-left languages can be a serious problem for developers and testers [10].

Last but not least, **data security** becomes a serious concern. Users share a great amount of private and personal information with apps, so they want to make sure that information cannot be accessed by unauthorized third parties. All this information and data needs to be secured and kept safe from hacking attempts.

4. MANUAL TESTING

Manual testing can be described as the part of [mobile] testing that requires human input, analysis, or evaluation [11]. Consequently, manual testing is a user-centric approach, mainly focusing on explorative ways of monitoring whether a mobile application meets user requirements and expectations.

Especially when it comes to the testing of the application's look and feel, the usability or the reproduction of bugs, manual testing on real devices is absolutely vital.

Furthermore, a manual testing approach is needed since not every device-specific function, such as location data or other environmental sensor data, can be automated. Therefore, running only automated tests is not sufficient and manual mobile app testing must be seen as an important component of QA testing [9].

In the following chapter, we will introduce the key challenges of manual testing.

MAIN CHALLENGES OF MANUAL TESTING

In many cases manual testing is **quite inefficient and time-consuming,** as live testers have to repeat different, but simple routines (e.g. login tests) on a variety of different OS platforms and end devices over and over again.

Another source of inefficiency is the lack of **productivity of live testers** (respectively the efficiency of devices) as they have to test an array of multiple devices. Consequently, this means that most of these devices remain unused, because testers can only test on one device at a time.

Device efficiency can also become a problem, when testers have to **share a single pool of devices.** Especially when they have to work across different locations, this could lead to coordination issues and unnecessary delays.

Furthermore, manual testing can be **error-prone**. As long as human testers are involved, misunderstandings are possible, particularly in situations in which multiple app versions must be tested and complexity multiplies [12].

Moreover, manual testing is often **conducted after the actual development.** This means that bugs are likely to be found at a very late stage of the development process, which leads to high costs of change (especially compared to an early detection of an error through continuous integration and regression testing).

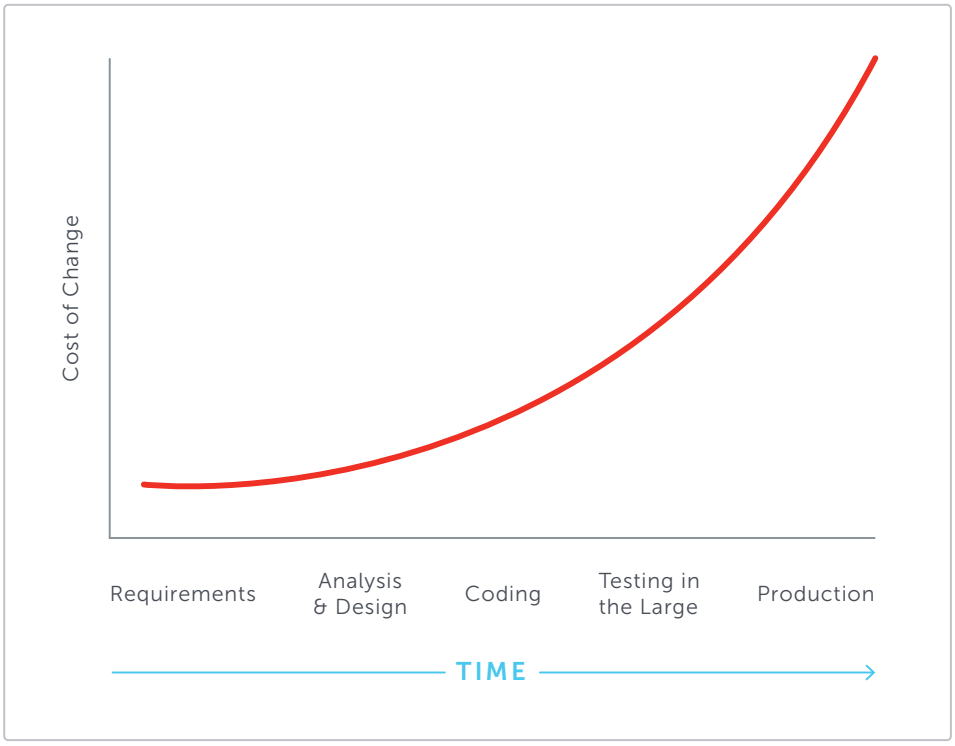


Figure 2: With manual testing bugs are likely to be found at a very late stage of the development process [13]

Testers taking mobile devices and removing them from the company's environment always carries a **potential security risk.** In doing so, viruses can be caught on infected pages or a new version of an OS can be downloaded accidentally. In addition, the risk that mobile devices are getting lost or stolen is much higher.

COSTS OF MANUAL TESTING

As stated above, manual testing often goes along with inefficiencies and heavy costs. These costs especially arise from:

- A high number of test devices, which must be bought or lent, to ensure adequate testing and to achieve certain coverage of different types (see: device fragmentation)

- Labor costs for the staff (group of live testers)
- Potential delays, which are caused by the shipping of devices
- Increased coordination and administrative efforts caused by the allocation of tested devices – especially among groups which are locally separated
- Eventual additional costs through loss of devices or security issues

5. AUTOMATED TESTING

Test automation can be described as the process of automating different test cases or scenarios by using a specific tool and language, instead of testing manually with a group of human testers and physical devices [14].

User Interface testing and continuous integration are two integral parts of test automation, which shall be introduced in the following. The second part of the chapter focuses on the advantages and disadvantages of automated testing.

UI TESTING FOR THE BEST USER EXPERIENCE

GUI (graphical user interface) testing ensures that an application renders the desired UI output in response to a sequence of user actions on a device. Regarding mobile app GUI testing, the tester analyzes the way an app handles different user input events and components, such as menu bars, toolbars, dialogs, buttons, edit fields, list controls and images.

As a black box and functional practice, GUI testing does not require testers to know the internal implementation details of the tested app. It only ensures that the user interface meets its written specifications and expected output when a user performs a specific action or enters a specific input.

Therefore, UI testing should never be skipped, as it tests specifically what users will visually experience on their devices. Implementing GUI testing at an early stage in the software development cycle speeds up the productivity of developers and testers, improves the code's quality and reduces the risks of finding bugs towards the end of the development cycle.

GUI testing is commonly approached manually by testers who run tests to verify that the app is behaving as expected. But as stated above, manual testing can be time-consuming, tedious, and error-prone. New methods of testing that have proved to be more efficient and reliable include automated UI testing using a software testing framework.

CONTINUOUS INTEGRATION /CONTINUOUS DELIVERY

A prominent practice that brings out the best in automated testing is continuous integration (CI) and continuous delivery (CD), a workflow that enables constant testing of each new build or change made in the source code [15]. CI/CD offer development and testing teams an agile development process with the ease of automated testing that is executed continuously each time a new update is presented to the software.

In traditional integration testing, which would be implemented as a waterfall process, testing would be done with two methods. The first one would be to let a sampling of users try out the application (crowd testing) once the writing process is over. In the second method, a number of test cases will be executed at a late stage of the software development process, usually after all development is done. CI/CD testing is flexible and executed at all stages of the software's development, starting as early as possible.

In practice, CI/CD orchestrate the way new code is introduced regularly by teams of developers to a shared mainline code. The CI/CD workflows start with a shared source code repository that team members use. With each significant change in the code, an automated build is conducted. When a team runs this cycle successfully, code is constantly developed and committed back to the mainline.

With the right CI/CD toolsets, testers and developers are able to automate and speed up the workflow by eliminating problems, which are often caused by long integration cycles. CI provides developers with a stress-free development environment as their software is being continuously tested every time new code is deployed. Testers can notify developers of any failed tests immediately, enabling quick updates, fixes, and iterations, resulting in continuous delivery.

Benefits:

- Developers concentrate on coding
- Faster iterations and release to the market
- Blind spots are eliminated by reducing the risk of broken code
- A bug free product is provided to the customers

BENEFITS AND DRAWBACKS OF AUTOMATED TESTING

Benefits:

The biggest benefit of automated testing is **time savings**, especially achieved through regression testing and continuous integration. Apart from regression testing, teams can also save time through the use and increased speed of tools that run tests much faster than human live testers do.

Another important advantage of automated testing is **cost reduction**. Since it requires fewer resources, test automation also reduces the overall costs involved in testing [14].

Furthermore, testers benefit from the **repeatability of tests** while using an automated approach. Consequently, tests can be re-run in exactly the same manner. This helps to avoid the risk of human errors such as testers forgetting their exact actions or missing out steps from the test script, which can result in either defects not being identified or the reporting of invalid bugs [14].

A powerful automated testing test suite helps to ensure that test scripts are kept up to date, and testers are able to **cover every feature (increased coverage)** within the application [14].

Finally, automated tests are **re-usable** and can be applied on different versions of the software – even if the interface changes [14].

In summary, test automation helps to save time and to reduce costs. The testing process and software quality can be increased through the reliability, repeatability and extent of the test suite. Finally, automation helps to increase the test coverage and to re-use these tests for different software versions and interfaces.

Drawbacks:

- The **initial effort** of automated mobile testing **seems** often **rather high** due to initial direct costs. But already after a few test executions the return on investment (ROI) increases rapidly and test automation pays off. This correlation is clearly illustrated in Stefan Münch's paper The Return on Investment (ROI) of Test Automation, where the following figure is taken from [16].
- Furthermore, automated testing **requires expert knowledge** especially when it comes to writing automated test scripts. Here, in-depth knowledge of the scripting languages of the used tool is mandatory.
- Not **every test can be automated** and not every project is suitable for automated testing, for example location data, and other environmental sensor data is really hard to test in a lab situation [9].

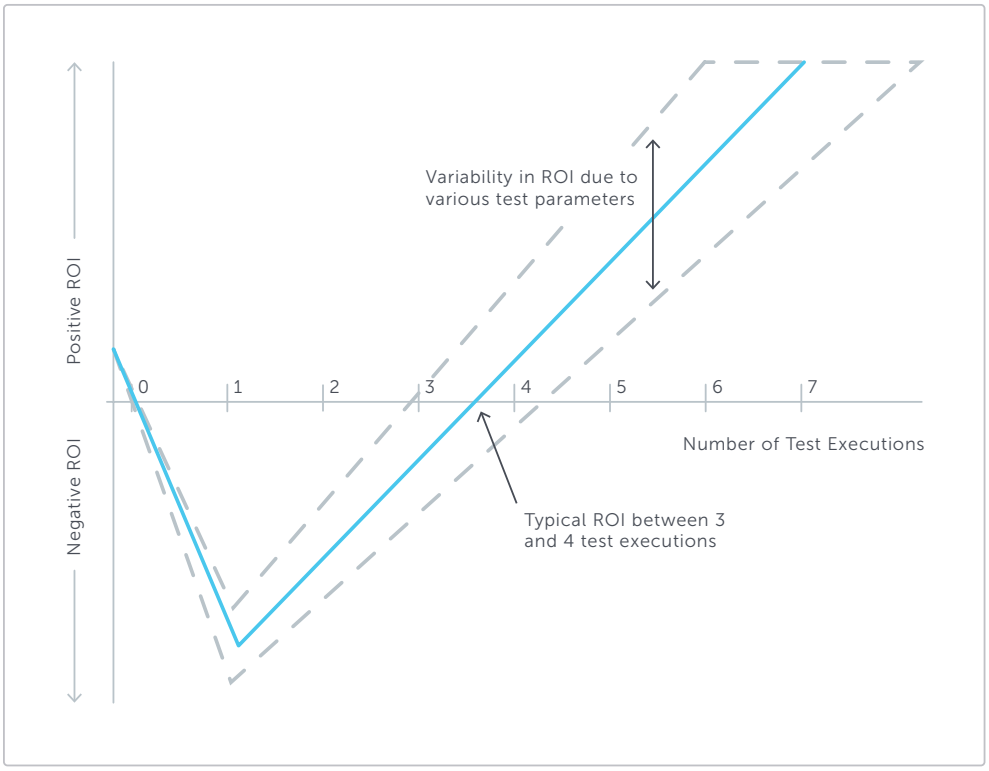
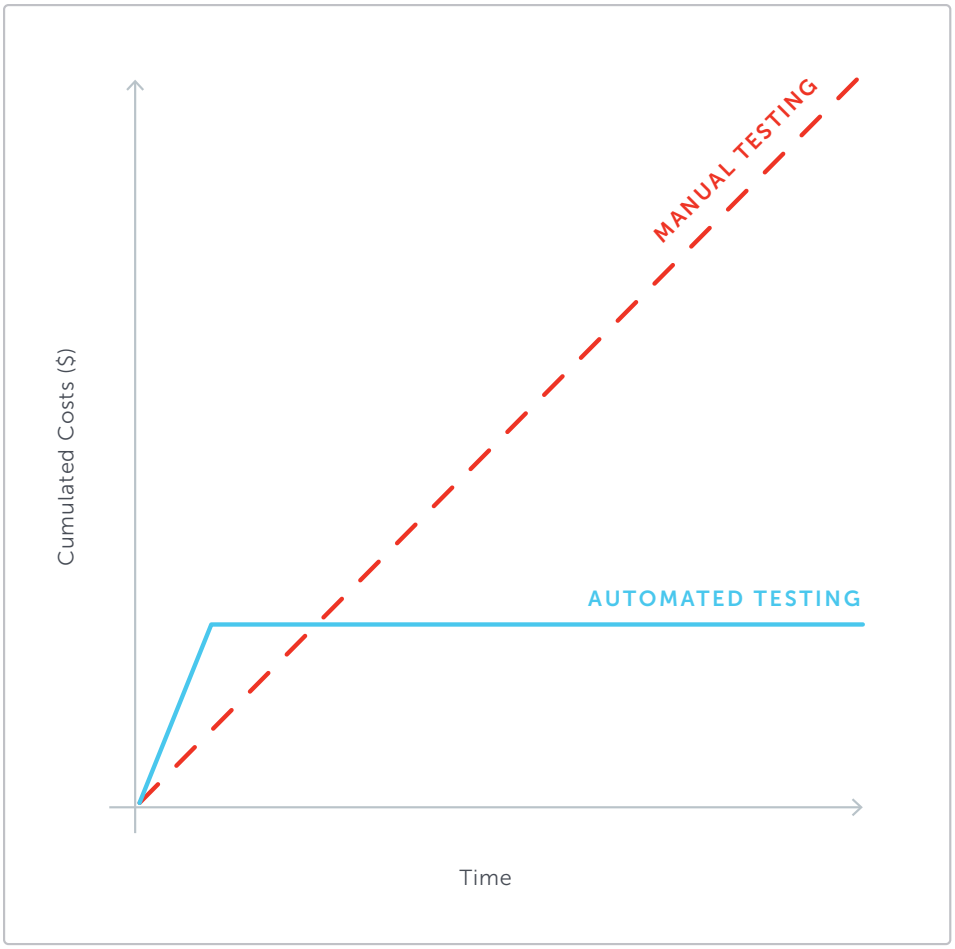


Figure 3: The typical ROI from the use of test automation tools [16].



CONCLUSION – MANUAL AND AUTOMATED TESTING

As described above, neither the exclusive performance of manual nor the exclusive performance of automated testing is sufficient to cover all test scenarios and mobile features. Therefore, a comprehensive mobile testing strategy requires both approaches to overcome the different challenges and limitations mentioned [9].

6. MOBILE AUTOMATION TESTING FRAMEWORKS

There are a number of great mobile testing frameworks that should be considered when defining an individual mobile testing strategy. All have different ways of tackling mobile automation and the right solution for a project will ultimately come down to which framework fits best to the specific needs of a user or customer [18]. Furthermore, many mobile testing frameworks are open source and supported by strong communities.

In the following section we will introduce and compare four different mobile testing frameworks.

6.1 APPIUM

Developed by Dan Cuellar, Appium is an open source test automation tool for native and hybrid mobile apps. Using JSON wire protocol and the Selenium WebDriver, it is compatible with both Android and iOS native apps, as well as hybrid and web apps.

In contrast to most other automation tools available, it does not require an extra agent, which needs to be compiled with the application code so that the tool can interact with the mobile app. This ensures that the tested app is in fact the app which will be submitted to the app store afterwards. Appium is an HTTP server written in node.js which creates and handles multiple WebDriver sessions for different platforms. One of the key principles of Appium is that test scripts can be written in any framework or language such as Ruby on Rails, C# and Java without having to modify the apps for automation purposes.

The fact that Appium is open source – with improvements and changes freely shared within the community – as well as the fact that it seamlessly runs on a variety of devices and simulators/emulators, makes it one of the best choices for mobile test automation [19].

Benefits:

- All complexities are under the hood of the Appium server – there are no restrictions towards programming language or the platform that is automated
- Supports cross-platform mobile testing – the same test can work on multiple platforms
- Does not require any extra agents
- Automation of web, hybrid and native mobile applications possible

6.2 CALABASH

Calabash is a cross-platform mobile test automation framework for native and hybrid Android and iOS apps, maintained by Xamarin. The framework enables automated UI Acceptance Tests written in Cucumber.

With the help of Cucumber, it is possible to express the behavior of the tested app using a natural language [9]. This approach is called Behavior Driven Development (BDD) and can be very helpful when business experts or non-technical colleagues are involved in the acceptance criteria process. Cucumber uses Gherkin as the Domain Specific language (DSL) to annotate the behavior of the application [9].

The actual test automation, however, is performed with the programming language Ruby and within so-called step definitions. In summary, Gherkin is responsible for describing the behavior of the application, Ruby is needed for the actual coding, and Cucumber is the framework that executes everything together on real devices or emulators. The framework's tests can be executed either from the command line, from an IDE or from a continuous integration server [9].

Benefits:

- Through the Behavior Driven Development (BDD) philosophy with Calabash, application behaviors are specified, instead of creating tests that describe the shape of APIs
- Cross-platform approach
- Supporting Android and iOS native apps

6.2 XCUITEST

This tool comes directly from Apple, created for developers. You can use this tool to test iOS native and hybrid apps from Xcode 7 IDE. XCUITest is a recorder that enables creating automated tests easily inside Xcode 7. With the latest update, you can take advantage of Appium Webdriver architecture to run XCUITest tests.

Tests can be written/recorded for real devices or emulators/simulators with either Objective C or Swift. This framework supports both app and device contexts to test apps comprehensively.

Benefits:

- Supported by Apple
- Integrated in Xcode IDE
- It's a recorder - easier for test writing
- Hybrid and native apps
- Languages: Objective C/Swift

6.3 ROBOTIUM

Another open source mobile testing framework is Robotium, which has full support for native and hybrid Android applications. Developed in 2010 by Renas Reda, this framework makes it easy to write powerful and robust automatic black box UI tests for applications. With the support of Robotium, test case developers can write function, system and user acceptance test scenarios, spanning multiple Android activities [21].

Regularly updated, Robotium also provides a so-called productivity tool – the Robotium Recorder. This commercial offering enables Android developers and testers to gather outputs and screenshots [21].

Benefits:

- Requires minimal knowledge of the application under test and minimal time to write solid test cases
- Test cases are more robust due to the run-time binding to UI components
- Fast test case execution
- Integrates smoothly with Maven, Gradle or Ant to run tests as part of continuous integration [21]

6.4 ESPRESSO

Open sourced by Google, Espresso is the latest Android test automation framework that enables developers and testers to run tests on x86 machines in the cloud and to apply their UI tests [22].

Espresso's core API is small, predictable, and easy to learn and yet remains open for customization. Without the distraction of boilerplate content, custom infrastructure, or messy implementation details, Espresso tests state expectations, interactions, and assertions clearly [23]. Being supported up to API level 25, Espresso allows its users to quickly write concise and reliable Android UI tests [23].

Benefits:

- Small, well-defined and predictable API, which is open to customization
- Seamless synchronization of test actions and assertions with the UI of the application under test
- Clear failure information – Espresso provides rich debugging information when a failure happens [24]
- Deep integration with Android

7. TESTING INFRASTRUCTURE

Mobile application testing is often conducted under different environmental and infrastructural conditions. These conditions range from a **nonprofessional infrastructure** with limited device capacities and without the use of CI/CD, to **professional infrastructural conditions** with private or shared cloud, enabling the use of test automation and continuous integration/ continuous delivery.

Irrespective of professional or nonprofessional testing conditions, all testers have to do a selection of "top devices" to ensure a best possible coverage of their target group.

In this context Daniel Knott recommends a selection of the top 10 to 15 devices used by (...)[the] target group in different variations [9]. Here, especially variations including smaller and older device types are very helpful to also cover "statistical outliers".

After distinguishing between nonprofessional and professional testing infrastructure, the following chapter will introduce the benefits of both cloud and on-premise solutions.

NONPROFESSIONAL TESTING INFRASTRUCTURE

The term nonprofessional testing infrastructure refers to a testing environment with little device capacities and without standardized buying processes. In a nonprofessional environment, several teams have to share the same pool of devices, which leads to inefficiencies and higher costs. Furthermore, neither parallelization of test automation nor continuous integration can be applied. Consequently, automation cannot be integrated in the agile development process.

PROFESSIONAL TESTING INFRASTRUCTURE

Within a professional testing infrastructure, a sufficient number of test devices is available to run manual tests. In addition, a professional infrastructure contains private and public cloud solutions that offer the opportunity to access a variety of additional real devices directly from the customer's browser. Here, standard test cases can be automated and integrated into CI/CD processes.

CLOUD SOLUTIONS

A mobile test cloud gives testers the opportunity of accessing hundreds of simulators, emulators and real devices to run their automated and manual tests on. Test can also run in parallel with test automation to save testing time, while results can be recorded and illustrated through screenshots and videos.

In addition, cloud services support tests through the most popular automation platforms like Appium, Robotium and Espresso. The upfront investment can

vary depending on your needs, but it will save you a lot of money in the long run, because licensing and renewal costs are minimized, which reduces total cost of ownership. These are the main reasons that make the concept of a mobile testing cloud an attractive proposition for developers and testers [25].

Sauce Labs offers public and private cloud solutions to test on real devices, as well as a cloud to test on emulators and simulators. Any native, hybrid or web app can be uploaded and tested easily directly from your browser.

Public Cloud

Public cloud is a secure solution that enables manual and automated testing on over 200 iOS and Android devices. High concurrency is also one of the benefits of the private cloud, even though not all the devices that you want to test on might be available when you need them. Devices are thoroughly cleaned after each use, so no user's data will be left on the testing devices.

Private Cloud

Private cloud is designed for customers that need the highest security, as well as for those customers who test frequently and at higher concurrencies as part of a CI/CD delivery pipeline. Having devices at exclusive use 24/7 enables faster development and deployment. Being the sole users of dedicated devices will add an extra layer of security on top of a secure IP SEC VPN connection.

This solution will give you the option to use CI/CD at full capacity, deploying hourly or daily. A private cloud can be an addition to public cloud if the needed devices are not available in the public cloud.

NONPROFESSIONAL INFRASTRUCTURE	PROFESSIONAL INFRASTRUCTURE
Little device capacities	Sufficient number of test devices
No standardized buying processes	Standardized buying processes
Test automation cannot be parallelized	Parallelization of test automation possible
Automation cannot be integrated in the agile development	Standard test cases can be automated and integrated into CI/CD processes

Figure 5: Main characteristics of nonprofessional and professional testing infrastructures

CONCLUSION

In the ongoing development of an app, automated testing provides a safety net for both developers and testers. The daily test runs ensure that the core functionality is working properly, the overall stability and quality of the app is transparently reflected by the test statistics, and identified regressions can be easily correlated to recent changes.

Used in the right manner, testing can be a powerful tool in fighting against the fragmented mobile device landscape. The crucial component of an effective testing strategy is to define custom tailored test cases for the application at hand and define a workflow or process that streamlines testing.

As stated above, testing mobile apps is a major challenge, but it can be solved efficiently with a structured approach, the right set of tools, and expertise.

REFERENCES

1.) Statista, (2015). Number of apps available in leading app stores 2015 | Statistic. [online] Available at: http://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/ [Accessed 5 Jul. 2015].

2.) Smartbear.com, (2015). What Is Mobile Testing?. [online] Available at: http://smartbear.com/all-resources/articles/what-is-mobile-testing/ [Accessed 5 Jul. 2015].

3.) Veracode.com, (2015). Mobile Application Security | Veracode. [online] Available at: http://www.veracode.com/products/mobile-application-security [Accessed 5 Jul. 2015].

4.) TATA Consultancy Services, (2015). [online] Available at: http://www.tcs.com/SiteCollectionDocuments/White%20Papers/Mobility_Whitepaper_Mobile-Application-Testing_1012-1.pdf [Accessed 5 Jul. 2015].

5.) Mona Erfani Joorabchi. Ali Mesbah. Philippe Kruchten, (2013). Real Challenges in Mobile App Development. University of British Columbia. Vancouver, BC, Canada.

6.) Opensignal.com, (2015). Android Fragmentation Report August 2015. [online] Available at: https://opensignal.com/reports/2015/08/android-fragmentation/ [Accessed Feb. 3rd 2017].

7.) Rouse, M. (2012). What is mobile device fragmentation? - Definition from WhatIs.com. [online] Available at: http://searchconsumerization.techtarget.com/definition/mobile-device-fragmentation [Accessed 30 Jun. 2015].

8.) Rouse, M. (II) (2015). What is regression testing? - Definition from WhatIs.com. [online] SearchSoftwareQuality. Available at: http://searchsoftwarequality.techtarget.com/definition/regression-testing [Accessed 19 Jul. 2015].

9.) Knott, D. (2014). Hands-on mobile app testing.

10.) Testlio, (2014). 6 key challenges of mobile app testing - Testlio. [online] Available at: https://testlio.com/blog/post/6-key-challenges-of-mobile-app-testing [Accessed 30 Jun. 2015].

11.) Softwaretestingclass.com, (2015). What is Manual Testing | Goal of Manual Testing | Manual Testing types | Software Testing Class. [online]
Available at: http://www.softwaretestingclass.com/what-is-manual-testing/ [Accessed 19 Jul. 2015].

12.) Mobilelabsinc.com, (2015). Manual Mobile App Testing - Mobile Labs. [online]
Available at: http://mobilelabsinc.com/solutions/mobile-app-testing/manual-mobile-app-testing/ [Accessed 30 Jun. 2015].

13.) Agilemodeling.com, (2015). Examining the Agile Cost of Change Curve. [online] Available at: http://www.agilemodeling.com/essays/costOfChange.htm [Accessed 19 Jul. 2015].

14.) Red-badger.com, (2015). Benefits of Automated Testing - Red Badger - Red Badger. [online]
Available at: http://red-badger.com/blog/2013/02/01/benefits-of-automated-testing/
[Accessed 4 Jul. 2015].

15.) Rouse, M. (2015). What is continuous integration (CI)? - Definition from WhatIs.com. [online]
SearchSoftwareQuality.
Available at: http://searchsoftwarequality.techtarget.com/definition/continuous-integration
[Accessed 5 Jul. 2015].

16.) Stefan Münch et al. (2012). The Return on Investment (ROI) of Test Automation. In: Pharmaceutical Engineering, July/August 2012, Volume 32, Number 4.

17.) Methodsandtools.com, (2009). Metrics for Implementing Automated Software Testing. [online]
Available at: http://www.methodsandtools.com/archive/archive.php?id=94 [Accessed 4 Jul. 2015].

18.) Saucelabs.com, (2015). Mobile Testing Tools - 11 Open Source Frameworks Compared. [online] Available at: https://saucelabs.com/resources/mobile-testing-tools [Accessed 4 Jul. 2015].

19.) 3Pillar Global, (2013). Appium: A Cross-browser Mobile Automation Tool. [online] Available at: http://www.3pillarglobal.com/insights/appium-a-cross-browser-mobile-automation-tool [Accessed 4 Jul. 2015].

20.) Developer.xamarin.com, (2015). Introduction to Calabash - Xamarin. [online] Available at: http://developer.xamarin.com/guides/testcloud/calabash/introduction-to-calabash/ [Accessed 4 Jul. 2015].

21.) Code.google.com, (2015). robotium - The world's leading Android™ test automation framework - Google Project Hosting. [online] Available at: https://code.google.com/p/robotium/ [Accessed 5 Jul. 2015].

22.) Avram, A. (2015). Google Espresso: Fast Automated Android UI Testing in the Cloud. [online] InfoQ. Available at: http://www.infoq.com/news/2013/10/google-espresso-testing [Accessed 5 Jul. 2015].

23.) Code.google.com, (2013). Espresso - android-test-kit - a fun little Android UI test API - Google's Testing Tools For Android - Google Project Hosting. [online] Available at: https://code.google.com/p/android-test-kit/wiki/Espresso [Accessed 5 Jul. 2015].

24.) Stackoverflow.com. (2015). Google Espresso or Robotium. [online] Stackoverflow.com. Available at: http://stackoverflow.com/questions/20046021/google-espresso-or-robotium [Accessed 5 Jul. 2015].

25.) www.gfi.com, (2015). [online] Available at: https://www.gfi.com/whitepapers/Hybrid_Technology.pdf [Accessed 5 Jul. 2015].

26.) 10th annual state of agile report (https://versionone.com/pdf/VersionOne-10th-Annual-State-of-Agile-Report.pdf) [Accessed March, 1st, 2017]

SAUCELABS

ABOUT SAUCE LABS

Sauce Labs is the leading provider of continuous testing solutions that deliver digital confidence. The Sauce Labs Continuous Testing Cloud delivers a 360-degree view of a customer's application experience, ensuring that web and mobile applications look, function, and perform exactly as they should on every browser, OS, and device, every single time. Sauce Labs is a privately held company funded by Toba Capital, Salesforce Ventures, Centerview Capital Technology, IVP, Adams Street Partners and Riverwood Capital. For more information, please visit saucelabs.com.

saucelabs.com/signup/trial

FREE TRIAL

SAUCE LABS INC. - HQ 116 New Montgomery Street, 3rd Fl San Francisco, CA 94105 USA